NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A HYPERTEXT TRANSFER PROTOCOL SERVER ON A HIGH ASSURANCE MULTILEVEL SECURE PLATFORM

by

Evelyn Louise Bersack

December 2000

Thesis Advisor: Cynthia Irvine
Second Reader: Geoffrey Xie

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 4

20010215 017

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE : Implementation of a HyperText Transfer Protocol Server on a High Assurance Multilevel Secure Platform		5. FUNDING NUMBERS
6. AUTHOR(S) Evelyn Louise Bersack		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

ABSTRACT *(maximum 200 words)*

In a client/server environment on a local area network (LAN), a server should provide various network applications including a hypertext transfer protocol (HTTP) server. HTTP is a client/server, request/response application protocol that is used on the World Wide Web (WWW). It provides the definition and means for transferring objects across internets. A server used in the context of a multilevel secure (MLS) LAN should be no exception. A MLS LAN should be capable of providing an HTTP web server that can be used by commercially available web browsers executing on client workstations. This server needs to be aware of the MLS environment and provide clients access to all web pages and objects for which they are authorized.

This thesis implements an HTTP web server running on a high assurance host in a MLS LAN. The web server is based on a commercially available web server application. The commercially available application has been modified and configured to run on the high assurance host. This thesis discusses the details for implementing the web server on the high assurance host.

The result of this thesis is an HTTP web server application that runs on a high assurance host servicing clients on a MLS LAN that are using commercially available web browsers. These clients now have the capability of web browsing at varying levels of classification on one workstation.

14. SUBJECT TERMS Hypertext Transfer Protocol, Web Server, Multilevel Secure, Local Area Network, High Assurance	15. NUMBER OF PAGES 144
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPLEMENTATION OF A HYPERTEXT TRANSFER PROTOCOL SERVER ON A HIGH ASSURANCE MULTILEVEL SECURE PLATFORM

Evelyn Louise Bersack
Civilian, United States Army
B.S., University of Arizona, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2000

Author: [redacted]
Evelyn Louise Bersack

Approved by: [redacted]
Cynthia Irvine, Thesis Advisor

[redacted]
Geoffrey Xie, Second Reader

[redacted]
Dan Boger, Chairman
Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In a client/server environment on a local area network (LAN), a server should provide various network applications including a hypertext transfer protocol (HTTP) server. HTTP is a client/server, request/response application protocol that is used on the World Wide Web (WWW). It provides the definition and means for transferring objects across internets. A server used in the context of a multilevel secure (MLS) LAN should be no exception. A MLS LAN should be capable of providing an HTTP web server that can be used by commercially available web browsers executing on client workstations. This server needs to be aware of the MLS environment and provide clients access to all web pages and objects for which they are authorized.

This thesis implements an HTTP web server running on a high assurance host in a MLS LAN. The web server is based on a commercially available web server application. The commercially available application has been modified and configured to run on the high assurance host. This thesis discusses the details for implementing the web server on the high assurance host.

The result of this thesis is an HTTP web server application that runs on a high assurance host servicing clients on a MLS LAN that are using commercially available web browsers. These clients now have the capability of web browsing at varying levels of classification on one workstation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. PURPOSE 1
B. RESEARCH QUESTIONS 1
C. OVERVIEW 2
D. BENEFITS OF RESEARCH 3
E. ORGANIZATION OF THESIS 5

II. BACKGROUND 7
A. HYPERTEXT TRANSFER PROTOCOL 7
B. APACHE SOFTWARE FOUNDATION PRODUCT 14
C. XTS-300 PLATFORM 16
D. NPS MULTILEVEL SECURE LOCAL AREA NETWORK PROJECT 17

III. ANALYSIS OF THE APACHE SOFTWARE PACKAGE 23
A. IMPLEMENTATION REQUIREMENTS 23
B. IMPLEMENTATION DECISIONS 24
C. PROBLEMS AND DIFFICULTIES 29

IV. IMPLEMENTATION OF AN APACHE-BASED HTTP WEB SERVER ON THE XTS-300 COMPUTER 33
A. APACHE SOURCE DIRECTORY STRUCTURE 34
B. MAKEFILE MODIFICATIONS 36
C. PHASE ONE 37
D. PHASE TWO 46
E. PHASE THREE 47
F. PHASE FOUR 50
G. DOCUMENTATION MODIFICATIONS 51

V. SECURITY CONSIDERATIONS 53
A. DIRECTORY STRUCTURES AND FILE ACCESS 53
B. APACHE ADD-ON MODULES 53
C. SECURE SOCKET LAYER 59

VI. CONCLUSIONS AND FUTURE WORK 63
A. DISCUSSION 63
B. FUTURE WORK 64
C. CONCLUSIONS 65

APPENDIX A: GLOSSARY 67

APPENDIX B: APACHE SOFTWARE LICENSE FILE 71

APPENDIX C: DIRECTORY LISTING 73

APPENDIX D: MODIFICATIONS TO CONFIGURATION FILES 79

APPENDIX E: MODIFICATIONS TO SOURCE CODE 81
APPENDIX F: MODIFICATIONS TO HEADER FILES 95
APPENDIX G: TOP LEVEL MAKEFILE 97
APPENDIX H: SERVER CONFIGURATION FILE 101
APPENDIX I: APACHE MODULES 117
LIST OF REFERENCES 121
INITIAL DISTRIBUTION LIST 125

LIST OF FIGURES

Figure 1	HTTP Request Message	12
Figure 2	HTTP Response Message	13
Figure 3	MLS LAN Configuration	19
Figure 4	Basic LAN Architecture	20
Figure 5	Basic MLS LAN Architecture	21

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1 HTTP Methods 12
Table 2 HTTP Status Codes 14
Table 3 Renamed Apache Files 51
Table 4 Documentation Files Referencing Renamed Files 51

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I must thank my husband, Stephen, daughter, Samantha, and son, Patrick for all of their patience, understanding, help, and frequent fishing trips. Without their cooperation and understanding, my time at NPS would have been more difficult than it was. Thanks Steve, Samantha, and Patrick.

I also want to thank Mr. David Shifflet for all of his time and contributions to this project. He provided many hours of help and guidance. Without his knowledge and experience, the timing issue might not have been solved. Thank you Dave.

I finally want to thank my thesis advisor Dr. Cynthia Irvine. She provided much needed guidance, support, motivation, and wisdom. Thank you Dr. Irvine.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

In a client-server environment on a local area network, the server should support and provide various network applications such as File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Telnet (a remote login facility), and HyperText Transfer Protocol (HTTP). A server used in the context of multilevel secure (MLS) local area network (LAN) should be no exception and provide these same network applications. A MLS LAN should be able to provide network applications that are capable of running at multiple security levels based on the level of the session between the client and server.

By providing an HTTP application on a server in a MLS LAN, the server can now provide web-browsing capabilities to its clients running at different session levels. This would allow clients to use a web browser to access web pages, documentation, graphic images, and other information on the server at varying levels of security classifications. The HTTP application on the server must be aware of the MLS environment and be configured to know what level of classification and assurance the client is requesting. It should also be able to provide the correct level of access to its web pages and information.

B. RESEARCH QUESTIONS

Several research questions are addressed in this thesis. The majority of the questions deal directly with implementing an HTTP server on a high assurance platform, more specifically the Apache web server package [Ref. 1]. Access control to the web pages that are being served is also a concern. The negotiation of the contents of the pages

that the server can access is not investigated. For this implementation, the server should have no idea about the contents of the pages it is serving to its client but be concerned about whether it has access to the objects the client is requesting.

The following questions are specifically addressed in this thesis.

1. Can the "Apache" software package be modified to run on the Wang XTS-300?
2. How difficult will it be to port the existing "Apache" software onto the XTS-300 platform?
3. Which version of the "Apache" software should be used?
4. Will there be any side effects that would adversely affect enforcement of access control policies?
5. Will the evaluation rating or trustworthiness of the system be adversely affected by the addition of an HTTP server? Will additional trusted code be required to support the server?
6. Will the modified "Apache" software package be easily maintainable across future upgrades to the XTS-300 STOP Operating system and/or "Apache" software?

C. OVERVIEW

The purpose of this thesis is to implement and test an Apache-based HTTP Web Server application on a high assurance host operating in a Multilevel Secure (MLS) Local Area Network (LAN) operating at the Naval Postgraduate School (NPS) Center for

INFOSEC Studies and Research Laboratory. This application will be implemented using a software package called Apache. Apache is available free from the Apache Software Foundation and can be downloaded from the Apache Software Foundation web site at www.apache.org [Ref. 2]. The Apache package is to be implemented on a Wang XTS-300 computer running the STOP-4.4.2, UNIX-like operating system [Ref. 3]. The goal of this undertaking is to provide a low cost commercially available HTTP Web Server application that can be configured and easily maintained on the XTS-300 platform.

D. BENEFITS OF RESEARCH

In a typical environment that handles classified information, the information is stored on some type of removable storage device that is kept locked and secured in a safe. When access to the information is needed, the storage device is removed from the safe and used in an isolated environment. When the user is finished with the information, the storage device is returned to the safe. There must be strict documented access security policies and procedures in place in this type of environment to ensure the prevention of disclosure and/or penetration of the information. Occasionally, the policies and procedures are not followed, and the information is compromised.

A high assurance multilevel secure (MLS) workstation eliminates the need for this type of environment. Information at all levels of classification can reside on the same high assurance MLS workstation. The high assurance MLS workstation enforces a strict access control policy that prevents the unwarranted disclosure and penetration of information.

The Department of Defense (DoD) conducts and supports testing programs and maneuvers that produce test information at all levels of classification. If this information resides on a high assurance MLS workstation, it will be protected by the access control policies in place on that platform. A drawback with using a high assurance MLS workstation is that in order for users (Program Managers, Test Engineers, and customers) to access their information they must have direct access to the host. This is an impractical restriction to place on the users, but allowing them access to the information through a network and COTS web browser is practical. To support this COTS web browser, a HyperText Transfer Protocol (HTTP) web sever application must be available on the high assurance MLS host. This high assurance MLS host could then be configured to act as an HTTP server in a MLS LAN environment. The users would not need direct access to the host but only require a client workstation, equipped with a COTS web browsing application, configured on the MLS LAN. The HTTP server, constrained by the access control policies enforced by the high assurance MLS host, allows users access to information that is centrally located but available to only authorized and identified users. Information served through such an HTTP server in a MLS LAN environment would not be vulnerable to unwarranted penetration and disclosure. The MLS LAN provides the same degree of security and access control that the stand-alone MLS host provides while allowing users the advantage of using their workstation and COTS web browsing application.

This solution is more practical and beneficial. All levels of data and information reside on a MLS host preventing the need for separate storage devices at different

classification levels that must be kept locked in a safe. The access control policies are strictly enforced by the MLS host and are not left up to chance by relying on the user to follow documented procedures. It is practical because most offices have COTS workstations with COTS web browsing applications. Special "custom-built" applications and programs are not needed to access the information, just a COTS workstation enhanced to be a trusted client on the MLS LAN with a COTS web browsing application is required. This thesis aims to provide the HTTP server application hosted on the high assurance MLS workstation so this benefit can be achieved.

E. ORGANIZATION OF THESIS

The remainder of this thesis is organized as follows:

Chapter II provides background material on the HyperText Transfer Protocol, the Apache Software Foundation Apache Web Server Package, the XTS-300 Platform, and the NPS Multilevel Secure Local Area Network Project.

Chapter III discusses the design constraints and source code requirements needed to implement an Apache-based server on a host in a MLS LAN environment and how these requirements can be met. It also discusses some of the initial problems encountered while trying to satisfy these constraints and requirements.

Chapter IV outlines the phases used to implement the Apache-based server. It highlights the changes made to the configuration files that were used to create the *Makefiles* for the C language source code compilation. It details the parameters used in the platform-dependent configuration C language header file. It discusses the major

changes to the C language source code to enable the server to run in a MLS LAN environment.

Chapter V discusses some security considerations surrounding the Apache-based server. It describes some of the Apache add-on modules that are or could be considered security related. It provides a brief discussion on Secure Socket Layer (SSL) and its inclusion into an Apache-based server.

Chapter VI answers the questions asked in this chapter. It suggests some topics for future work related to this thesis.

There are several Appendices in this thesis. Appendix A is a glossary of terms and acronyms. Appendix B is a complete directory listing of files on the XTS-300. Appendix C is the Apache Software License File. Appendices D-H are Apache files that have been modified for this Apache-based server implementation. Appendix I provides a listing of Apache add-on modules provided with the Apache server software package.

II. BACKGROUND

This chapter provides background information on the HyperText Transfer Protocol (HTTP), the Apache Software Package, the Wang XTS-300, and the Naval Postgraduate School (NPS) Multilevel Secure (MLS) Local Area Network (LAN) Project.

A. HYPERTEXT TRANSFER PROTOCOL

The HyperText Transfer Protocol (HTTP) is a client/server, request/response application protocol that runs on top of TCP (Transmission Control Protocol). TCP is an internet transport protocol that is connection-oriented and has a reliable data transfer service as well as congestion control mechanism [Ref. 2]. The application that most widely uses HTTP is the World Wide Web (WWW). HTTP has been used in the web since 1990 and has now gained the position as the most used protocol on the Internet [Ref. 4]. The protocol defines the format of the messages that can be passed between a client, usually a web browser such as Netscape Communicator or Microsoft Internet Explorer, and a server such as Apache, Microsoft Internet Information Server, or Netscape Enterprise Server.

There are several key terms used with HTTP. The following are some of the terms used within this overview.

1. Cache: A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. [Ref. 5]

2. Client: An application program that establishes connections for the purpose of sending requests. [Ref. 5]

3. Connection: A transport layer virtual circuit established between two application programs for the purpose of communication. [Ref. 5]

4. Entity: The information transferred as the payload of a request or a response. [Ref. 5]

5. Message: The basic unit of HTTP communication, consisting of a structured sequence of octets transmitted via the connection. [Ref. 5]

6. Object: A file, such as a HyperText Markup Language (HTML) file, a JPEG image, a GIF image, a Java applet, and/or an audio clip. [Ref. 2]

7. Request: An HTTP request message. [Ref. 5]

8. Resource: A network data object or service, which can be identified by a Uniform Resource Identifier (URI). [Ref. 5]

9. Response: An HTTP response message. [Ref. 5]

10. Server: An application program that accepts connections in order to service requests by sending back responses. [Ref. 5]

11. URI: Uniform resource identifier. Formatted strings that identify, via name, location, or any other characteristic, a resource. [Ref. 5] (Defined in IETF RFC 2396 [Ref. 6].)

12. URL: Uniform resource locator. Formatted strings that identify the location of an object. (Defined in IETF RFC 2396 [Ref. 6].)

13. User Agent: The client that initiates a request.

HTTP was developed to help simplify the way users access information on the Internet. HTTP is generic in nature. It can be used to transfer ASCII text, hypertext, audio, images, and any other Internet accessible information or data. The information that can be transferred using HTTP is flexible, both in content and length. HTTP is a stateless protocol in that the server keeps no state information about the client's activities or requests. However there is an option, called "cookies" that can be used to allow the server to keep some information about the client and user. There are currently two versions of HTTP that most current servers and clients support simultaneously. HTTP 1.0 was proposed in Internet Engineering Task Force (IETF) Request for Comment (RFC) 1945 [Ref. 7] as an informational RFC. HTTP 1.1, currently the standard, was first proposed in IETF RFC 2068 [Ref. 8] and later clarified in IETF RFC 2616 [Ref. 5]. The IETF has several RFCs that are related to HTTP including RFC 2145 that discusses the use of the HTTP version numbers [Ref. 9] and RFC 2295 that discusses "Transparent Content Negotiation with HTTP 1.1" [Ref. 10]. Another HTTP related RFC is RFC 2617, "HTTP Authentication: Basic and Digest Access Authentication" [Ref. 11]. This proposal improves on the security methods for authenticating users to a server and outlines methods for authenticating users to a server without sending passwords in clear text form (the method used in HTTP 1.0).

HTTP 1.0 was simple in nature. The client initiated a TCP connection, the server accepted the connection, the client sent a request, the server serviced the request and then closed the connection. One major disadvantage to HTTP 1.0 was that, by default, the TCP connection was closed after each transfer of an object. A web page might be

compromised of several objects that include hypertext and images. HTTP 1.0 would require that a TCP connection be established for each object on the page. For example, if a page contained one hypertext file and six graphic images, seven separate TCP connections would need to be established. This connection establishment/teardown is very inefficient. It adds overhead traffic to the network, which contributes to network congestion. It is also slower because each connection requires the initial establishment. A partial remedy was proposed in an effort to overcome this limitation but was proven to be relatively ineffective. HTTP 1.0 was also found to have several bugs and shortcomings. These problems lead to the proposal of HTTP 1.1.

HTTP 1.1 was first proposed in IETF RFC 2068 [Ref. 8] and clarified in IETF RFC 2616 [Ref. 5]. These RFCs attempt to make HTTP a well-behaved, well-defined, generic Internet protocol. The proposal clearly specifies that HTTP 1.1 must be compatible with HTTP 0.9 and HTTP 1.0. This backward compatibility allows for ease of migration and upgrading of client/server applications that were initially designed for HTTP 1.0. One area of improvement was the TCP connection state. HTTP 1.1 allows persistent TCP connections. This greatly improves the efficiency of the protocol. A connection may be used for one or more message/object exchanges. The requests/responses can be pipelined without waiting for each response to be serviced, so that a single TCP connection can be used more efficiently and with much lower elapsed time. These persistent connections also come with a drawback. The issue as to when to close the TCP connection must now be considered. Three methods have been specified. The client can explicitly request to close the connection using a *Close* connection header

field option within the message. The server can issue a *Close* connection response using a header field option informing the client that the connection will be closed. The server can time-out on an inactive connection and then close the connection.

HTTP 1.1 provides for virtual hosts allowing service providers to assign multiple domain names to a single IP address. The server will be able to distinguish the pages from the domain name used in the URL. This is accomplished using the *Host* header field in the request message [Ref. 12]. This is a major change from HTTP 1.0 in that this *Host* header field must be present in the message so that the server can correctly service the request.

HTTP 1.1 also improves the caching capabilities of clients and servers. It provides well-defined rules and a caching model, which allows both servers and clients to control the level of cachability and the conditions under which the cache should update its contents [Ref. 12].

There are two main types of HTTP messages that both HTTP 1.0 and HTTP 1.1 support, the *request* message and the *response* message. The client usually sends the request message to the server. The server responds to a client's request with a response message.

The first line of the request message is called the request line. It has three distinct fields: the method field, the Uniform Resource Locator (URL) field, and the HTTP version field. The method field is a key word that indicates to the server what action is being requested. The URL field gives information about the location of the object being requested. The version field is the version of HTTP the client is using [Ref. 2].

Following the request line are header lines. Most header lines are optional. Some header lines provide information about the client and what type of information the client is capable of handling. These header lines help the server do its job better by allowing the server to send objects to the client that it can handle and interpret. One header line that **must** be present with HTTP version 1.1 is the *Host* header line. If the requested URI does not include an Internet host name for the service being requested, then the *Host* header field **must** be given with an empty value [Ref. 13]. This line specifies the host on which the object resides. It allows for virtual hosts (see preceding paragraphs for explanation). Following the request line and header lines, an empty line followed only by a carriage return, CR, and line feed, LF, is sent to indicate the end of the request message.

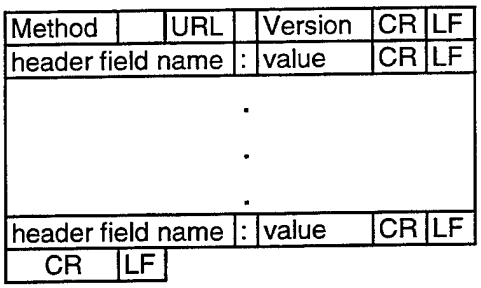

Figure 1 HTTP Request Message After Ref. [14]

Method	Definition	Version
GET	Get a header and resource from the server	1.0
HEAD	Return just the header, no resource	1.0
POST	Send information to the server	1.0
OPTIONS	Return the list of methods allowed by the server	1.1
TRACE	Trace a request to see what the server sees	1.1
DELETE	Delete a resource on the server	1.1
PUT	Create or change a file on the server	1.1
CONNECT	Enables proxies to switch to a tunneling mode for protocols like SSL	1.1

Table 1 HTTP Methods

The response message has three sections: a status line, header lines, and then the entity body. The first line of the response message is called the status line. It has three distinct fields: the protocol version field, a status code, and a corresponding status message. The version field is the version of HTTP the server is using. The status code and corresponding message indicate the result of the request from the client [Ref. 2]. Following the request line are header lines. Some header lines provide information about the server, connection information and general information. Other header lines will give information about the object being sent to the client such as the type of object being sent and the length of the object so that the client knows how to handle the object and how much data to expect. Following the status line and header lines, an empty line followed only by a carriage return, CR, and line feed, LF, is sent to indicate the beginning of the object. The object is then sent.

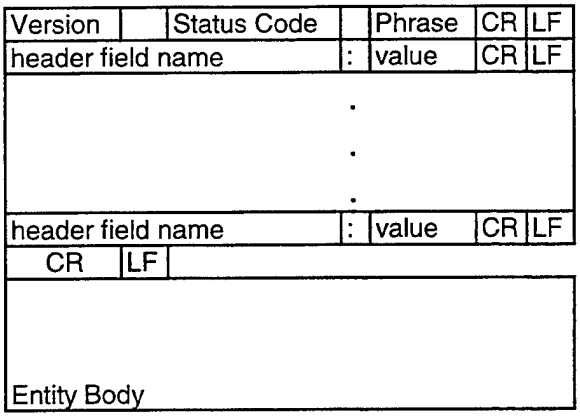

Figure 2 HTTP Response Message After Ref. [14]

Code Range	Meaning
100-199	Informational
200-299	Client request successful
300-399	Client request redirected, further action necessary
400-499	Client request incomplete
500-599	Server errors

Table 2 HTTP Status Codes

B. APACHE SOFTWARE FOUNDATION PRODUCT

The Apache Software Foundation has information about its activities and products at its web site, www.apache.org [Ref. 2].

The Apache HTTP server project is a collaborative software development effort aimed at creating a robust commercial-grade, featureful, and freely available source code implementation of an HTTP (Web) server [Ref. 2].

The Apache software version 1.3.12 was downloaded from the Apache Software Foundation site. Included in this package is C language source code for the server application, Readme files, *Makefiles*, configuration files, HTML documentation for the server, example source code for creating and adding platform/application dependent modules and information to help port, configure, and manage an Apache-based server. The application has already been successfully ported to many different platforms and all of these ports are freely available. The openness of Apache's source code is one of the major reasons for its popularity [Ref. 15].

The Apache software is covered by a license allowing its distribution (see Appendix B). The Apache Software Foundation provides no formal support for the Apache server software. The information available in their documentation as well as on

their web site [Ref. 2] is useful. In addition, there are also several web sites dedicated to the support of sites with Apache-based servers. There are numerous reference books available that provide varying levels of detail of information on Apache. The range of topics for reference books include configuration, maintenance, and optimization of Apache, to adding custom built modules, to securing an Apache-based server.

Apache is designed to work on a network. On a Unix platform, it can be activated as a daemon, running in the background waiting for a client application to request service [Ref. 15]. It can also be activated through an application that services network connections.

Apache is setup through configuration files. These files contain Apache directives that control the server's behavior. Using the configuration file makes Apache extremely versatile and gives the administrator comprehensive control over the features and security provided by Apache [Ref. 15].

The Apache Software Foundation package provides many additional modules that administrators can configure into the server to meet their needs. To add a module to an Apache-based server, the module must be compiled and built with the server. It must then be configured with the correct directives in the server configuration file. The ease of adding and configuring add-on modules allows administrators to fine-tune an Apache-based server and only configure the modules they require. There is also a large amount of third party modules available for Apache servers that can provide more capabilities and flexibility [Ref. 15]. A list of the modules provided with the Apache source code package is presented in Appendix I.

It is this flexibility, coupled with Apache's stability and performance, and the availability of its source code that makes it the most popular choice of web server software on the Internet [Ref. 15].

C. XTS-300 PLATFORM

The XTS-300 product is a combination of STOP 4.4.2, a multilevel secure operating system, and a Wang Government Services Inc. supplied x86 hardware base [Ref. 16]. The current hardware platform is Pentium-based. STOP is a Unix-like multiprogramming operating system designed not only to support much of the Unix System V interface for applications software but to produce and run object programs that adhere to a subset of the "Intel 386 Family Binary Compatibility Specification 2" as well [Ref. 16].

The XTS-300 provides network connectivity in the evaluated configuration, through TCP/IP and single level Ethernet built into the Trusted Computing Base (TCB). However, it does not provide network application servers [Ref. 16].

The XTS-300 provides Mandatory Access Control (MAC) that allows for enforcement of both a security and an integrity policy. Enforcement of the mandatory confidentiality policy is based on a mechanism that meets the requirements formally expressed in the Bell and LaPadula security model [Ref. 17]. Mandatory integrity policy enforcement is based on a mechanism that adheres to the Biba integrity policy model [Ref. 18]. The system implements Discretionary Access Control (DAC) and provides for user identification and authentication needed for user ID-based policy enforcement [Ref. 16].

Implementation of a Secure Attention Key (SAK) provides a trusted path mechanism. The system enforces the "principle of least privilege", giving users no more authorization than that required to perform their functions [Ref. 16]. The TCB makes use of hardware features to provide process separation and TCB isolation. The TCB has been designed and implemented to resist penetration [Ref. 16].

The XTS-300, STOP 4.4.2, was evaluated against the Department of Defense (DoD) TCSEC [Ref. 19] and rated at level Class B3. This Class B3 rating implies not only incorporation of particular security features but also a high level of assurance [Ref. 16].

D. NPS MULTILEVEL SECURE LOCAL AREA NETWORK PROJECT

The Naval Postgraduate School (NPS) Multilevel Secure (MLS) Local Area Network (LAN) project's goal is to provide true multilevel access to information over a LAN [Ref. 20]. This is an ongoing project at the NPS Center for INFOSEC Studies and Research Department.

There are three main components that comprise the MLS LAN: a high assurance multilevel secure server, a Trusted Computing Base Extension (TCBE) with a Trusted Application Protocol Server (TPS), and a Secure Session Server that resides on the high assurance server.

The first component is the high assurance multilevel secure server platform. The server must be equipped with a Trusted Computing Base (TCB) which provides a penetration resistant security enforcement mechanism for the MLS LAN operations [Ref. 20] [Ref. 21] [Ref. 22]. The TCB will provide the mandatory and discretionary access

controls to information as well as user identification, authentication, and authorization services [Ref. 21] [Ref. 22]. The Wang XTS-300 acts as the high assurance server in the MLS LAN. It is equipped with a TCB that enforces the security policies on the protocol application servers.

The second component in the MLS LAN is the Trusted Computing Base Extension (TCBE) [Ref. 23]. The TCBE has three main functions: establish a trusted path between the user and the high assurance workstation, enforce the object reuse provision of the TCB, and enforce access control policies [Ref. 22]. The TCBE is required because commercial-off-the-shelf (COTS) workstations running COTS operating systems and applications are not trusted or trustworthy. The COTS workstations are equipped with a TCBE that will provide a trusted path to the high assurance server. The client workstation used in the MLS LAN is a Pentium-based personal computer (PC) running Microsoft Windows NT.

The TCBE in place for this implementation was software driven. A software application was provided that allowed the user to establish a trusted path with the high assurance server. A trusted software application, the Trusted Path Server (TPS) (sometimes referred to as the TCBE server) is used to communicate with the client and provide the trusted path [Ref. 24]. This communication connection is made over Ethernet. The TPS is responsible for accepting user logins and session level negotiations for the client workstations. It is also responsible for associating a username and session level with each login from the client workstation.

The third component of the MLS LAN is a software program called Secure Session Server (SSS) [Ref. 24]. This is a trusted application on the high assurance server that acts much like the Unix "inetd" daemon. It accepts protocol requests on various port numbers and creates a process to handle the communication with the client. For each accepted protocol request, the SSS receives the user ID and session level, provided through the TPS, associated with the client workstation. The SSS uses this information to create the protocol server to handle the client's request. The server executes as the user ID and at the session level of that user. The data is passed back and forth from the client to the protocol server via the SSS.

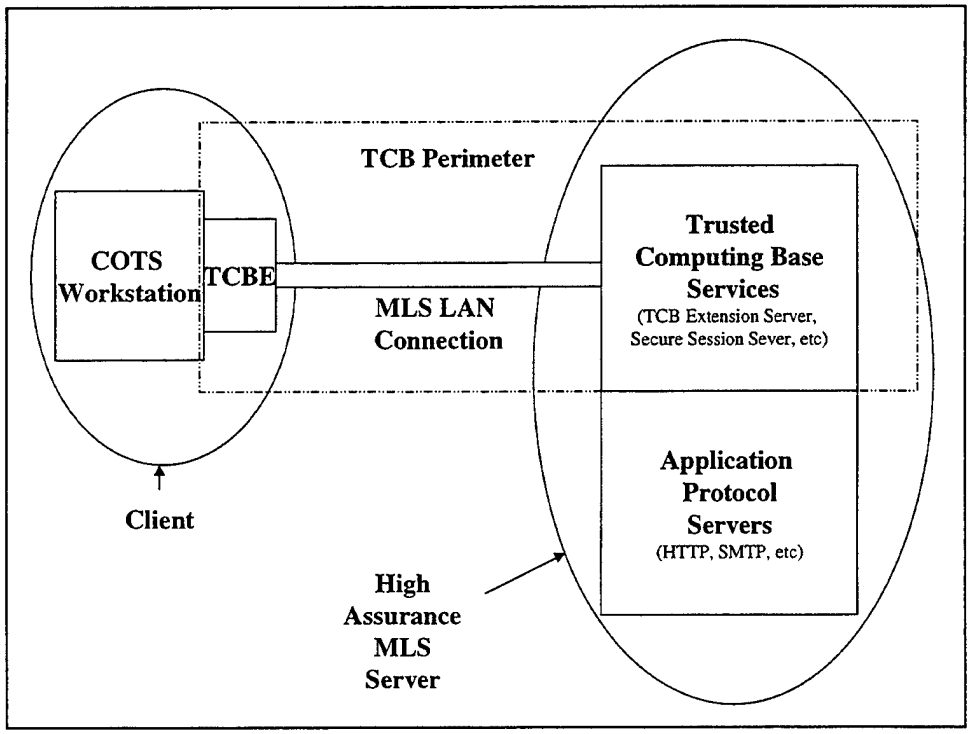

Figure 3 MLS LAN Configuration After Ref. [22]

The protocol server and the Secure Session Server communicate through a "pseudo socket". The pseudo socket can be described as the communication medium, or a

virtual socket, or the read/write buffer between the two processes. Its only function is to pass data between the two processes.

The two figures below represent a basic LAN Architecture and a MLS LAN Architecture. A basic LAN has no trusted components and cannot provide high assurance access control to multilevel information. This information is vulnerable to attacks and disclosure from any client that has access to the LAN.

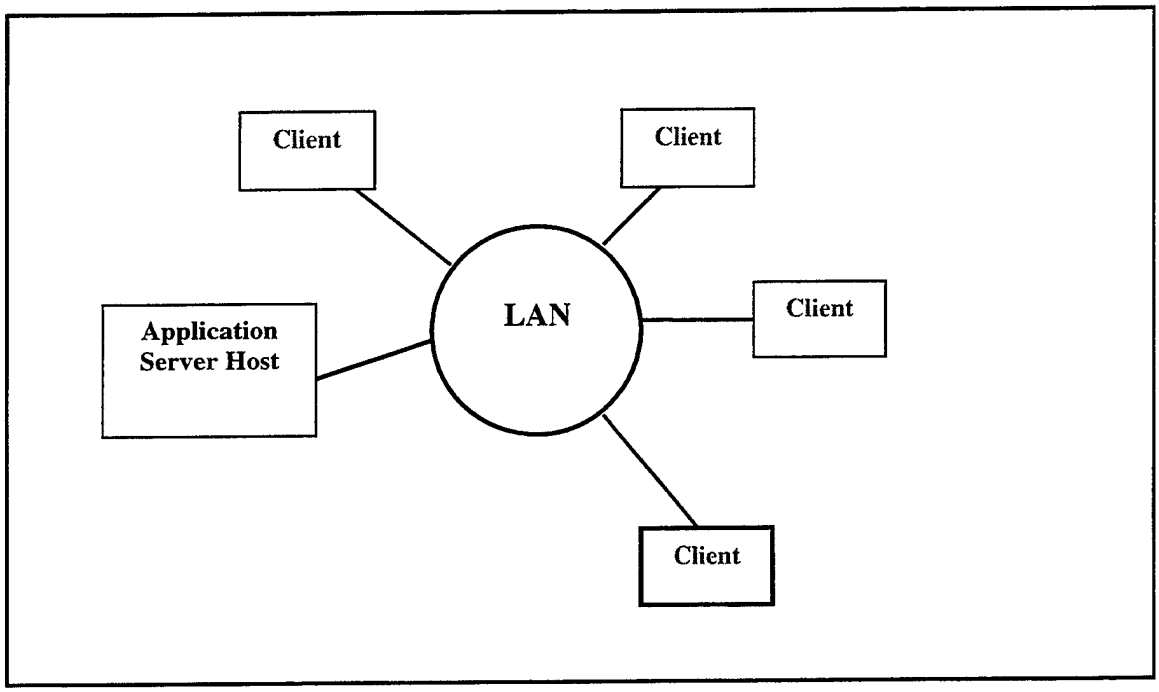

Figure 4 Basic LAN Architecture

With a MLS LAN Architecture, there are trusted and high assurance components that provide access control and protection to the multilevel information. The trusted architecture includes the high assurance host and clients enhanced with the TCBEs. In the MLS LAN, once a client-server connection is established (a trusted path), other clients (trusted or not) cannot penetrate the trusted connection. Information cannot be attacked or

disclosed through the trusted path and the high assurance server enforces access control policies on application servers servicing client requests.

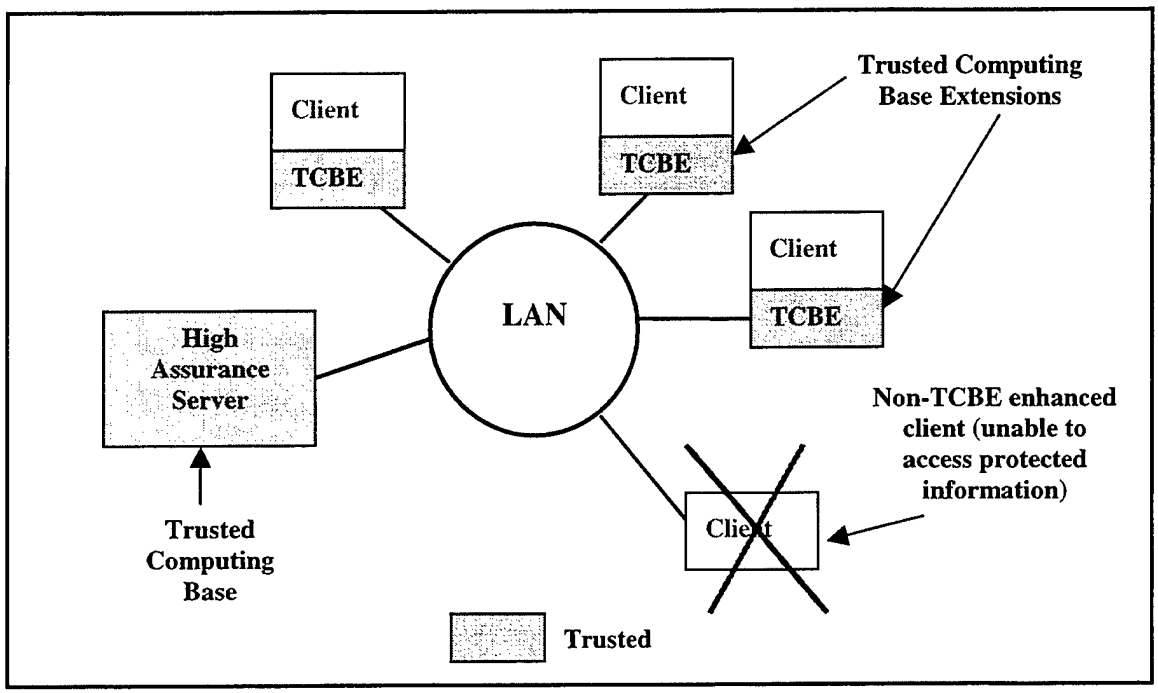

Figure 5 Basic MLS LAN Architecture After Ref. [23]

THIS PAGE INTENTIONALLY LEFT BLANK

III. ANALYSIS OF THE APACHE SOFTWARE PACKAGE

A. IMPLEMENTATION REQUIREMENTS

This Apache-based server was implemented to function in a MLS LAN environment. A MLS LAN server application implemented on the XTS-300 is accessed through a Trusted Computing Base Extension, TCBE. The XTS-300 MLS LAN applications are the Trusted Path Server, TPS, and Secure Session Server, SSS. The XTS-300 would do all authentication of the user and session level. If a request for an HTTP service comes through the TCB tunnel, the SSS would activate the HTTP server application. All information passed between the client and server would pass through this protected trusted path. The communication between the server and the client would be accomplished through "pseudo sockets" rather than network sockets. The pseudo socket will be the communication path between the server application and the SSS. This pseudo socket communication imposed several requirements and restrictions on the HTTP server.

1. The server must be activated by another application namely the Secure Session Server (SSS).
2. The server must be stand-alone; it should not spawn or create children or additional processes.
3. The implementation calls for no parameters or arguments to be passed to the server application.
4. The server must communicate (both read and write) to the Secure Session Server through the pseudo sockets rather than network sockets.

5. The Secure Session Server must be configured to recognize and correctly respond to HTTP client requests.

6. The server must not adversely affect the evaluation rating of the system.

7. The server should be permitted read-down privileges, allowing users at a higher level read access to lower level objects.

8. The server should not be permitted read-up privileges, not allowing users at a lower level read access to higher level objects.

The rest of this chapter addresses these requirements and concludes with a section on the difficulties and problems encountered during the porting of the Apache-based server and what solutions were attempted to solve these problems.

B. IMPLEMENTATION DECISIONS

1. The server must be activated by another application namely the Secure Session Server (SSS).

This requirement was accomplished through modification of the server configuration file. One of the Apache directives is *ServerType*. The directive controls the way in which Apache handles multiple copies of itself as well as how Apache is activated. The arguments are *inetd* or *standalone* (default value). By using inetd, the server knows that it has been activated by another process. On a Unix platform, the inetd daemon receives requests for network applications through port numbers. The inetd daemon will then activate the corresponding server and hand off the connection to the server to allow it to service the request.

2. *The server must be stand-alone; it should not spawn or create children or additional processes.*

This requirement was accomplished through the *ServerType* directive. By setting the value to *inetd*, another process servicing network connections (on a Unix platform this is the inetd daemon) is responsible for activating individual servers each time an HTTP request comes in. The server will exit once it has finished servicing the request. The inetd option will be used on the XTS-300 platform to allow the SSS to activate one server at a time. The Apache Group has declared this method of activation as clumsy and inefficient [Ref. 25]. The Apache server was designed to be a server daemon that runs all the time and handles the network connections itself. The basic concept of the Apache server is that the server has several child servers. The parent server waits for a request and then hands the request off to a child or spawns another child to handle the request, allowing the parent to service another request. This method is fast; the parent quickly services the request by passing it to a child and is ready for the next request to come it. It also uses memory efficiently since the server configuration file is read at initial activation and the parameters are cached in the parent server's memory where they can be shared with all the children. By using the inetd option, each client request activates a new server specifically for that client. This results in a slower response time since the server must read the server configuration file and open any log files before it is ready to service the request. Because it cannot share memory with other servers, memory is used inefficiently.

The multilevel secure environment will take full advantage of these two conditions resulting from using the inetd option. The memory of a server running at one

classification level can not be shared with a server running at a different classification level. All log files must also be kept at different levels, thus preventing write-down. The inetd option supports these restrictions. Without the inetd option, the child servers are trusted with respect to MAC policies and untrusted with respect to DAC policies. This is unaccepted. The servers should be untrusted in both areas to allow the TCB to enforce the access control policies. The inetd option allows all application servers, activated as children of the Secure Session Server, to be untrusted with respect to MAC and DAC policies.

3. *The implementation calls for no parameters or arguments to be passed to the server.*

This was a problem since the server needs to know where to find the main configuration file. This path information is usually passed as an argument to the process. If using the inetd option and the inetd daemon on a Unix platform, you can specify options and arguments to pass to the application in the inetd.conf file. The SSS does not have this capability. The solution was to hardcode the configuration file path name in a configuration header file that is included with the Apache source code. The *ap_config.h* header file allows numerous platform-dependent parameter and variable definitions. This header file is to be modified based on the platform that Apache will be built on. The following line was added under the "XTS" block of definitions

#define SERVER_CONFIG_FILE /usr2/bersack/http/conf/httpd.conf

This definition is intended to be used as a constant. Once the configuration file is in place it should not be renamed or moved. The renaming or moving of the configuration

file will require a complete recompilation of the server. This definition could and should be removed once the Secure Session Server is configured to pass arguments to server applications, acting like the Unix inetd daemon.

4. *The server must be able to communicate (both read and write) to the Secure Session Server through the pseudo sockets rather than network sockets.*

This involved searching through the Apache C language source code to isolate the system-level calls that wrote to and read from the network sockets. Once these calls were identified, they were modified to call functions that communicate with the pseudo socket instead. These pseudo socket functions had been written and implemented in other network applications ported to this platform. Also, the network socket setup code had to be isolated and modified so that a network socket was not opened for communication, but redirected to communicate with the pseudo socket. To minimize changes to the Apache source code, the network socket structure was filled with data for the pseudo socket. This eliminated the need to find all references to the network socket variable and change it to the pseudo socket variable.

5. *The Secure Session Server must be configured to recognize and correctly respond to HTTP client requests.*

The Secure Session Server (SSS) reads a configuration file that has the following format:

Protocol ID Port # APS Path

The Protocol ID represents the protocol name, for HTTP it would be HTTP. The Port # is the network port number that the daemon should listen to for HTTP traffic. The default port number for HTTP is 80. This is a well-known port number and used by all commercially available web browsers. The APS Path is the full pathname for the application program server (APS). For this version of the Apache-based server the full pathname is /usr2/bersack/apache13/src/httpd. By adding the following line to the SSS configuration file, the SSS will be able to activate the server for any HTTP requests arriving on port 80.

HTTP 80 /usr2/bersack/apache13/src/httpd

This line must be changed and/or modified if the server application is moved or renamed or if the port number the server can service changes.

6. The server must not adversely affect the evaluation rating of the system.

The Apache-based HTTP server runs on the XTS-300. It communicates through the Secure Session Server (SSS). It is an untrusted application and does not enforce a security policy. It does not add to the Trusted Computing Base (TCB) on the XTS. For this reason, the software package should have no effect on the evaluation rating of the system. There are basic security issues concerning any Apache-based server hosted on any platform that must be addressed. These issues are discussed in the Apache documentation. Some security issues related to an Apache-based server in a MLS LAN environment are discussed in Chapter V.

7. *The server should be permitted read-down privileges, allowing users at a higher level read access to lower level objects.*

The server does not enforce this policy. The XTS-300 will enforce the access control policies. The server is activated at the session level the user is running at, with the permissions allowed for that user. The XTS-300 enforces the access control policies pertaining to the user ID and current session level. The Apache-based server will be constrained by the TCB. This will be true for objects embedded in a web page that the server tries to access as well as for the web page itself. If the object is at a lower level, the server will have access to the lower level objects but not to higher level objects. If the XTS-300 allows the server access to an object, the server will send that object to the client. If the XTS-300 does not allow the server access to an object, the server will not be able to serve the object to the client and will send an error message instead.

8. *The server should not be permitted read-up privileges, not allowing users at a lower level read access to higher level objects.*

The discussion for requirement 7 above applies to this requirement also.

C. PROBLEMS AND DIFFICULTIES

Several problems were encountered during this porting project. The first problem was encountered after the Apache-provided files were installed on the XTS-300. All of the text files had extra control characters at the end of each line. A generic shell script was written that would remove these control characters from the files and save the fixed file. This later led to problems with the image files that came as part of the Apache software package. All of the control characters were removed from these image files. This

corrupted the files. The client could not display the image, since it was corrupt. Once this problem was isolated, all of the image files were replaced and the client was able to correctly display them.

The next problem was to determine all of the various parameters and definitions to set in the configuration header file, *ap_config.h*, for the XTS-300 platform. A majority of the C language #define statements in the configuration header file are used to define or undefine system calls and functions specific to a particular target platform. The Apache documentation on porting defines a few of these parameters, but there are many others that were discovered as the porting progressed. A few of the parameters were incorrectly defined or used with default values. These caused numerous problems during the testing stage. The full list of parameters and variables that were set and used for the XTS-300 platform are discussed in the next chapter and provided in Appendix F.

The XTS-300 platform has very primitive debugging capabilities. Debugging was accomplished by adding print statements to the source code. This in itself caused several problems that included debug file creation, file access, and writing to the file.

A problem arose while testing the initial build of the server. An attempt was made to use a Telnet session on the local host machine to communicate with the server. This was tested before any pseudo socket code had been added and with minimal changes to the base Apache source code. Had the server been configured correctly, it should have sent a response message to the request message sent using the Telnet session. The Telnet session seemed to talk to the server and produce output to the error log file when errors were detected on the server side. However, the Telnet session did not produce the

expected response output to the window the Telnet session was running in. It was determined that the Telnet session was not echoing the characters from the response message to the window. To remedy this problem, a terminal *echo* program was used instead of the Telnet session. This program did provide the expected response headers and messages and indicated that the server was responding and working correctly.

Several other minor problems were encountered during the project. They are discussed in the next chapter, as they arose during the testing of the server.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION OF AN APACHE-BASED HTTP WEB SERVER ON THE XTS-300 COMPUTER

All modifications, additions, and changes made for this Apache-based implementation are presented in this chapter. It first describes the directory structure organization of the Apache files residing on the XTS-300 platform. It highlights the changes made to the configuration files that were used to create the *Makefiles* for the C language source code compilation. The parameters used in the platform-dependent configuration C language header file are described. It discusses the major changes to the C language source code to enable the server to run in a MLS LAN environment. The phases used to implement the Apache-based server are outlined.

The Apache-based web server was implemented in phases. The first phase concentrated on getting the server compiled with minimal changes to the base Apache C language source code. Once the server was compiled, the server was tested on the local host to ensure it was functioning as expected.

The next phase was to modify the C language source code to allow the server to communicate through pseudo sockets rather than network sockets. Once these modifications were finished, the server was tested in a client/server environment in the MLS LAN to isolate and correct problems.

The final phase was to test the server at different session levels, with different clients, and with different user personas. Problems arose during this phase that required modifications to the server configuration file.

We begin with a description of the Apache source directories and Makefiles.

A. APACHE SOURCE DIRECTORY STRUCTURE

The Apache-provided files were organized in a hierarchical tree directory structure. The top-level directory contains files describing basic Apache information, installation guidance, licensing agreement, and PGP keys for the Apache developers. It has five subdirectories: *cgi-bin*, *conf*, *htdocs*, *icons*, and *src*.

The *cgi-bin* directory contains sample test cgi-scripts to use if the mod_cgi module is used.

The *conf* directory contains sample server configuration files that should be modified for the target platform.

The *htdocs* directory contains HTML pages about the Apache server. The Apache manual is under this directory and contains valuable information to help implement and configure an Apache-based server. The manual documents are the same documents found at the Apache web site [Ref. 2].

The *icons* directory contains image files used with the Apache HTML documentation and manual.

The *src* directory contains all the files needed to compile and link the Apache server. It contains porting instructions, installation and configuration scripts, template Makefile, and general Readme instruction files. The top-level *Makefile* resides in this *src* directory. There are several subdirectories under the *src* directory that contain the C language source code and header files for the Apache server. These subdirectories are *ap*,

helpers, *include*, *lib*, *main*, *modules*, *os*, *regex*, and *support*. Each of these subdirectories contain a *Makefile* that is used to compile the source code in the directory.

The *src/ap* directory contains source code files that have various Apache system defined functions that can be used if the target platform does not support a particular function.

The *src/helpers* directory contains various scripts that are executed when the top-level *Configure* script is executed. These scripts help determine the operating system of the target platform and various characteristics of that operating system.

The *src/include* directory contains all of the Apache-provided C language header files required for the compilation of the Apache server.

The *src/lib* directory contains a subdirectory, *expat-lite* that provides source code used to parse XML files.

The *src/main* directory contains the core C language source code for the Apache server. Most of the modifications made for this implementation were to C language source code files in this directory.

The *src/modules* directory contains subdirectories of source code for Apache-provided add-on modules. These subdirectories are *example*, *experimental*, *extra*, *proxy*, and *standard*. The *example* directory contains example modules to help programmers with the Apache API and module concept. The *experimental* directory contains work-in-progress modules. The *extra* directory is provided to hold third-party modules. The *proxy* directory contains the C language source code for the Apache proxy module. The

standard directory contains the C language source code for the standard Apache-provided modules.

The *src/os* directory contains subdirectories that have C language source code related to a specific operating system. These subdirectories are *bs2000*, *mpeix*, *netware*, *os*, *os390*, *tpf*, *unix*, and *win32*. The *src/os/unix* directory was used in this implementation for the XTS-300 platform.

The *src/regex* directory provides C language source code to parse regular expressions (wildcards, ranges, the Unix ~) used in directives in the server configuration file.

The *src/support* directory contains example files and scripts that provide support for operating system functions such as log rotation and starting/stopping the server.

A complete listing of all files under the top-level directory is provided in Appendix C.

B. MAKEFILE MODIFICATIONS

The Unix "make" program is a software engineering tool. It aids in the development of large programs by keeping track of which portions of the entire program have been changed, compiling only those parts that have changed since the last compile. The make program relies on a file that provides rules and instructions for it to execute. The default filename for the make file is *Makefile*. Apache uses this default convention.

The Apache-provided *Configure* script dynamically builds all of the *Makefiles* using the options and parameters defined for the platform in the *Configure* file. The top

level *Makefile* and all lower level *Makefiles* are created this way even though template *Makefiles* are supplied with the source code. The Apache documentation recommends modification of the *Configure* script, rather than manual modification of *Makefiles*. This is suggested since there are several *Makefiles* created for the subdirectories when the *Configure* script is executed and using the *Configure* script ensures that all the *Makefiles* have the updated parameters needed for the specific target platform. The top-level *Makefile* has calls to lower level *Makefiles* that are needed to build the Apache-based server. A complete listing of the top-level *Makefile* is provided in Appendix G.

C. PHASE ONE

This phase concentrated on compiling, linking, and testing the server on the XTS-300 platform. The main goal was to get the Apache source compiled and linked, with as few modifications as necessary, so that it could be executed on the XTS-300 platform.

The initial step in this phase was to determine the version of Apache source code to implement. This step involved downloading a version, loading it onto the XTS-300, compiling and linking it, and ensuring that the required tools to support the configuration of the Apache server were available on the XTS-300 platform.

When this project was first undertaken, the Apache Software Foundation had just released Apache version 2.03a. This version was downloaded from the Apache Software Foundation web site and installed on the XTS-300 computer. The documentation for this version stated that two configuration management tools were needed in support of building the Apache server. These were "libtool 1.3" and "autoconf 2.13". Each had to be built and installed on the platform that Apache was to be built on. These two applications

are freely available GNU software applications. They were downloaded from the GNU web site [Ref. 26] and installed on the XTS-300. After some modifications to their *Makefiles* (needed for the XTS-300 platform), they were both compiled and linked.

At this point, the first attempt to build Apache was attempted. However, the Apache configuration file did not seem to recognize the presence of libtool and autoconf. The Apache documentation was unclear as to where these two tools had to reside in relation to the source code and configuration file; i.e. in the same directory as the configuration file, in a bin directory, or as a user-specified full path name within the server directory structure. Several configuration attempts were tried, but the Apache configuration script was still unable to recognize that these applications were present on the system. Rather than play a potentially endless guessing game, an alternative was chosen.

Documentation for Apache 1.3.12 contains no references to autoconfig or libtool, so it was decided to try this version of the software. This decision had several additional advantages. At the time of this port, the 1.3.12 version was the latest stable version of Apache. The 2.0* versions were being updated almost monthly on the web site with bug fixes and enhancements. Most of the known bugs in the 1.3.12 version are documented and fixes are available. Also the 1.3.12 package includes template *Makefiles* that do not rely on autoconfig or libtool. The project could concentrate on the Apache server rather than on tools that the server depended on to be built.

The second step was to use the Apache provided *Configure* script to create the platform dependent *Makefiles*. This script called several other provided scripts. One script

checked for location and presence of C language header files on the target system so that the include statements could be generated in the dynamically created *ap_config_auto.h* file.

The first script the *Configure* script called was the *GuessOS* script to determine the operating system to configure for. The *GuessOS* script used the Unix "uname" command with different options to try to determine the platform, operating system, and operating system version number if applicable. Once this information is determined, the script sets default parameters that the *Configure* script would use to generate the *Makefiles*. The *GuessOS* script could not interpret the XTS-300 STOP operating system because it was not a predefined case in the script. To remedy this situation, the following modifications were added to the *GuessOS* file:

```
##############################
#
# ADDED FOR XTS-300
#
##############################
    *:stop4.*:STOP:*)
        echo "${SYSTEM}-xts300-stop4.4"; exit 0
        ;;
##############################
```

The "SYSTEM" variable is the host name. The line returned from this script for this implementation is *holmes-xts300-stop4.4*.

Once the *GuessOS* script was modified to recognize the XTS-300 STOP operating system and return this information to the *Configure* script, the *Configure* script had to be modified to use the returned information correctly. The *Configure* script has many case statements that switch on the parameters returned from the *GuessOS* script. These case

statements are used to define and set parameters used in the *Makefiles*. All of the *Makefiles* are generated using this *Configure* script. The following lines were initially added to the *Configure* script file:

```
      ##########################
      #
      # ADDED FOR XTS-300
      #
      ##########################
         *-xts300-stop4.*)
             OS='XTS'
             CFLAGS="$CFLAGS -g -DXTS -
I/usr2/shifflet/wip/include "
             LDFLAGS="$LDFLAGS -lcass -lmw -lsocket -lgen"
             LIBS="$LIBS -L/lib -lsocket -lmw -lgen -
L/usr/lib -lcrypt -L/usr2/shifflet/wip/lib -lut_cass -lcass"
             RANLIB=true
             LN=ln
             ;;
      ##########################
```

The "OS" variable defines the platform the server was built for. The "CFLAGS" are used as compile options and defines the parameter "XTS". This parameter was used to isolate added code in the C language source files and headers to distinguish that the code was added for this implementation. The other variables are basic C language *Makefile* parameters used to define specific pathnames and options to use for this platform.

The *Configure* script also uses the *Configuration* file to control which add-on modules should be included and compiled. Execution of the *Configure* script generates a generic C language source code file called *modules.c*. The *Configuration* file contains directives for which add-on modules to compile with the Apache server. Modules are included for compilation by uncommenting the appropriate line corresponding to the module in the *Configuration* file. The format of the line is an "AddModule" directive

followed by the *path/mod_name* of object code of the module. The *Configure* script would then try to match the *mod_name* to a C language source file in the directory specified by *path.* If the *Configure* script could not find a matching C language source file, it would look at the *module* definitions within the C language source files in the directory to try to find a match. The *Configure* script only inspected the first word of the *module* definition. Some *module* definitions had two or three word definitions. When none of the first-word definitions matched the parameter the *Configure* script was searching for, the *Configure* script assumed that the add-on module could not be found. This produced an error message when the *Configure* script was executed. By changing the *Configure* script to inspect the entire first line in the *module* definitions, the parameter could be matched and no error messages were generated. Two lines were modified in the *Configure* script to look at the entire first line of a *module.* The modified lines with changes in bold are:

```
if ls -lt $file Configuration.tmpl | line | \
modname=`egrep '^module .*;' $modbase.c | line |\
```

Once these changes were complete, the *Configure* script could be executed. It produced the top-level *Makefile* as well as all subdirectory *Makefiles.*

The next step was to attempt to compile the unmodified source code. The compilation depends upon the C language *ap_config.h* header file that is used to define platform specific calls, functions, definitions, and parameters. It is also used to undefine default values predefined for all platforms. Compilation errors were expected on this first attempt since nothing had been added to this header file for the XTS platform. These

errors helped define the platform-dependent parameters and variables required in the C language header file *ap_config.h*. The error messages also indicated functions that could not be found in the system C language libraries and inconsistencies with variables defined in the system C language header files. After a lot of investigation into the Apache documentation, the XTS-300 programming manual, and documentation on other projects that ported applications to the XTS-300, the following list of parameters were added to the *ap_config.h* file to specify the "XTS" definitions (A complete final version of all modifications is provided in Appendix F.) :

```
/*************************
*
* ADDED FOR XTS
*
*************************/
#elif defined(XTS)

#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <setjmp.h>
#include <time.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/signal.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <sys/stat.h>
#define NEED_INITGROUPS  /* need groups */
#undef HAVE_MMAP         /* do not have mmap() */
#define HAVE_SHMGET      /* do support shmget() */
#undef HAVE_GMTOFF       /* no gmtoffset function */
#undef HAVE_CRYPT_H      /* no crypt.h header file */
#undef HAVE_SYS_RESOURCE_H    /* no sys/resource.h header
file */
#undef USE_MMAP_SCOREBOARD    /* do not use mmap */
#undef USE_SHMGET_SCOREBOARD  /* do not use shmget
scoreboard */
```

```
#define USE_LONGJMP       /* use the long jump functions */
#define USE_FCNTL_SERIALIZED_ACCEPT /* use this option for
fcntl() */
#define NO_MMAP           /* do not mmap() */
#define NO_KILLPG         /* do not have killpg() */
#define NO_SETSID         /* do not have setsid() */
#define NO_USE_SIGACTION /* do not use sigaction() */
#define NO_LINGCLOSE      /* do not allow linger close of
connections */
#define NO_GETTIMEOFDAY  /* system gettimeofday only wants
one parameter */
#define NEED_DIFFTIME
#define S_ISLNK(mode) 0  /* no symbolic links set to 0
(false) */
typedef int pid_t;
#define lstat stat        /* redefine lstat to stat */

#define STDIN_FILENO  0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
#define SIGCHLD SIGCLD    /* redefine SIGCHLD to SIGCHD */
```

The *#include* statements define XTS system C language header files to be included in files when compiled. The parameter "pid_t" had to be type-cast as an integer. This variable, which describes a process ID on Unix platform, was not defined in any of the XTS system header files so it had to be created. The other *#define* and *#undef* statements are used to indicate if certain functions are available on the underlying platform or if the server should use functions that are provided as part of the Apache server source code package. The Apache documentation explains many of these *#define* parameters and suggests default values for them. Determination of which parameters that were ultimately defined or undefined resulted from searching the XTS-300 programming manual to see if the function was supported by the XTS-300.

Once the *ap_config.h* file was modified another compilation was attempted. There were far fewer errors this time. The compiler complained about a *getpass()* function in the *ap_getpass.c* file. The function was not properly defined for the XTS platform. A return data type was not correctly typecast for the XTS system *getpass()* function. The following changes in bold were made to the *ap_getpass()* function in *ap_getpass.c*:

```
#if defined (XTS)
    (int) pw_got = getpass(prompt);
#else
     pw_got = getpass(prompt);
#endif
```

Another similar complaint was reported with an *initgroup()* function in *util.c* file. The XTS does not support this supplemental group function. The function had a provision for platforms that do not support this option so the flag was set for the XTS. The following change in bold was made to the *initgroup()* function in *util.c:*

```
    #if defined(QNX) || defined(MPE) || defined(BEOS) ||
defined(TPF) || defined(__TANDEM) || defined(NETWARE) ||
defined(XTS)
```

Another modification was associated with the *fcntl()* system call. The XTS system function *fcntl()* expected a different data type of parameter passed to it than what was being used in the Apache source code. Two lines were modified in the *http_main.c* file:

```
while ((ret = fcntl(lock_fd, F_SETLKW, (int)&lock_it)) < 0
&& errno == EINTR) {
while ((ret = fcntl(lock_fd, F_SETLKW, (int)&unlock_it)) < 0
&& errno == EINTR) {
```

After all of these modifications, the server was compiled and an executable was created.

The server was then tested in a standalone mode. When the server executes, it reads a server configuration file called *httpd.conf* whose pathname is passed as an argument to the program. The server configuration file required several changes to define attributes specific to the XTS and server directories. The following changes were made to the server configuration file *httpd.conf* (A complete listing of the *httpd.conf* file is provided in Appendix H):

```
ServerRoot /usr2/bersack/http

Port 2000

#User nobody
User bersack
#Group #-1
Group other

ServerAdmin bersack@holmes

ServerName 131.120.10.99

DocumentRoot "/usr2/bersack/http/htdocs"

LogLevel debug
```

The server was now activated as a *standalone ServerType* listening for traffic to arrive on port 2000. To communicate with the server, a Telnet session on the local host was attempted. The problem with this approach was discussed in the Chapter III-C. A terminal *echo* program was used instead. This program allowed the user to specify the port to communicate on and the message to send. Several lines could be sent to the server before sending the termination sequence: an empty line followed by a CR and LF. The server responded to each request. If the URL was not correct, an error message was logged to the error file and an error response message was sent. If the server could service

the request, the object was sent to the terminal echo program and displayed. These experiments showed that the server was working correctly.

D. PHASE TWO

This phase concentrated on making the server communicate through pseudo sockets, as described in Chapter II-D, rather than network sockets. This involved modifications to source code, the *ap_config.h* header file, the *Configure* script, and the server configuration file *httpd.conf*.

The first step of this phase was to identify and isolate the Apache system read and write calls that communicated with the network sockets. This proved to be very challenging. Several debug statements were added to various functions to help track the flow of the source code. Two functions were finally found in the *buff.c* source file called *ap_read()* and *ap_write()*. The comments for these functions claimed they were the lowest level read and write functions, calling the XTS system read() and system write() functions. The *ap_read()* and *ap_write()* functions were modified to use macro definitions in place of calls to the system read() and write() functions. The macro definitions are defined in the *ap_config.h* header file. If the functions were compiled with the "USE_P_SOCKET" (this definition was added to the *Configure* script to be added to all of the *Makefiles*), the macro definitions would reference the pseudo socket functions. If this option was not defined for the compilation, the macro definitions would reference the system-defined functions. This allowed the server to be compiled to use a pseudo socket or a network socket. The only modification required for this switch is the inclusion or absence of the definition of the "USE_P_SOCKET" parameter in the *Makefiles*.

The macro definitions were added to the *ap_config.h* header file. A full listing of the additions to the *ap_config.h* header file is provided in Appendix F.

The file *http_main.c* source file was modified to add the pseudo socket code. It was also modified to create and open a debug file that was used to help track down problems.

The final modification to be made in the *http_main.c* source code file was ensure the variable used for communication to the network socket referenced the information for the pseudo socket. This centralized the code and was more efficient than tracking down all references to the network socket variable and changing it to reference a pseudo socket variable. A complete listing of modifications in *http_main.c* is provided in Appendix E.

After all modifications and additions were complete, old compiled object code, libraries, and the executable file were removed from the system using the "clean" option in the *Makefile*. This was done to ensure a complete recompiling and linking of all source code. The server was compiled and linked and an executable file was created.

The server configuration file was modified so that the *ServerType* was set to *inetd*. Also the Trusted Path Server configuration file was modified so that it would activate the HTTP server.

E. PHASE THREE

The third phase of the server port was the testing phase.

Testing the server uncovered operational problems. The first major concern was whether the server was correctly and completely transmitting large files. The small files

would be completely transferred but large files would not. The buffer size for the Secure Session Server (SSS) communication was set to transfer data in chunks of 4096 bytes but did not keep track of how much data had been sent. Since the objects the server could send were of various size, the server needed to keep track of how much data it had sent to the SSS and how much it had left to send. The read/write functions that communicated with the SSS in *http_main.c* were modified to keep track of the amount of data sent and the amount of data waiting to be sent.

Even after this modification, all data in the files was not sent completely. Debug statements were added to the source code to follow the flow of the program. The problem was tracked to an Apache function that called a platform-dependent system function called *writev()*. The XTS-300 does not support this function. After some research into the Apache documentation, a *#define* parameter called "NO_WRITEV" was discovered. This parameter needs to be defined if a platform does not support the *writev()* function. This definition was added to the *ap_config.h* file and the server was recompiled. The server now served the pages completely.

Another problem encountered was with image files. The first inclination was to blame the image problem on the *writev()* problem, but once this was corrected the image files were still not correctly displayed by the client. After much debugging and searching through Apache handlers and C language source code, the image files on the XTS-300 were determined to be corrupt (see previous chapter). Once new images files were loaded onto the XTS-300, they were transferred and displayed by the client. The only modification needed for this problem was good image files.

The hardest and most challenging problem was related to the debug file creation. At times the server would be activated, create the debug file without writing to it, and exit without servicing the client request. Other times, if the debug file existed, the server would work properly. Even when the file was created *a priori*, successful service was hit and miss. Sometimes the server would service the request and other times it would just exit. This problem was very frustrating because no errors were being written to the error log and the information in the debug file gave no clues as to what the problem could be. The debug log would show that the server would get the request but never respond to it. After a lot of trial and error, and frustration, the problem was finally isolated to a timing problem. The SSS was activating the server and immediately signaling that it had data for the server to read. Sometimes the server was not ready to handle this signal and would just exit when the signal occurred. Other times it would service the signal but not have time to respond. To solve the problem, the SSS was modified so that it activated the server and then waited for the server response that it was ready for data. The code in *http_main.c* was modified so that it had time to create and open the debug file, setup the pseudo socket and then respond that it was ready for data. This solved the hit and miss service problem as well as the file creation problem.

The same type of behavior was occurring on the close of the connection. The SSS would close its communication connection with the server before the server could issue the close-connection response. This resulted in the client waiting for more data to be transferred from the server. The client had no indication that the data transfer was

complete. This was solved by having the SSS inform the client that the connection was to be closed.

The server now seemed to be working correctly in the MLS LAN environment.

F. PHASE FOUR

The final phase was to try the server at different session levels. The first attempt to request that the server serve a page at a high classification level resulted in no service. The error log file could not be created at this level in the current directory it was directed to write to. As a solution, the server configuration file, *httpd.conf*, was modified to place all server write-able files in the */tmp* directory on the XTS-300. This directory exists at all levels. Once this change was made, the server worked at all levels and with different user personas.

The *mod_include* module was added to the server and configured in the server configuration file to attempt to have the server serve a dynamic web page comprised of objects at different levels. An HTML file was created that used the include command to try and include files that were at different classification levels. In theory, the low session level server should not be able to access the higher level objects and report with an error message. The higher level server would have access to these lower level objects. The server responded correctly, serving only objects that the XTS TCB allowed access to. At the lower level, the high level objects were not included and not sent to the client. At the higher level, all lower level objects were included and sent to the client. This showed that the XTS TCB was correctly enforcing the access control policies on the server.

The server implementation on the XTS-300 platform in the MLS LAN was now complete.

G. DOCUMENTATION MODIFICATIONS

The maximum length of a filename on the XTS-300 is twenty-three characters. While transferring the Apache-provided files from a floppy disk to the XTS-300, some of the documentation file names were too long for the XTS-300. Thus some files had to be renamed. The table below shows a list of files that were renamed on the XTS-300 platform.

Original File Name	Renamed as
conf/highperformance.conf-dist	conf/highperf.conf-dist
htdocs/manual/content-negotiation.html	htdocs/manual/content-neg.html
htdocs/misc/known_client_problems.html	htdocs/misc/known_problems.html

Table 3 Renamed Apache Files

These files were referenced in the following documentation files:

File Name	Referenced (in htdocs/manual/)
content-negotiation.html	index.html new_features_1_0.html mod/core.html mod/mod_autoindex.html mod/mod_mime.html mod/mod_negotiation.html misc/FAQ.html misc/custom_errordocs.html
known_client_problems.html	env.html index.html misc/compat_notes.html misc/index.html

Table 4 Documentation Files Referencing Renamed Files

These documentation files were modified to reference the renamed files. Note that the file *highperformance.conf.dist* is an example file provided to help set up a configuration file for the server. It was not referenced in any of the documentation files.

V. SECURITY CONSIDERATIONS

A. DIRECTORY STRUCTURES AND FILE ACCESS

The Apache Software Foundation Apache Security Tips web site has a lot of useful information for the basic installation of the Apache server and the web pages it serves [Ref. 27]. It suggests that all files, directories, and parent directories, on a Unix system, be owned by root and be write-able by root only. This is also suggested for the log files and log file directories [Ref. 27]. It is suggested that the server be only read/write-able by root, thus preventing any user from replacing the executable, but executable by all. In a MLS LAN environment, it is not practical for the log files to be restricted this way. The server is activated by the SSS as a user, so the user needs at least write permission to the log files and read permission for the server configuration files.

Apache-based servers also use *.htaccess* files to read in user-defined directives. Users can configure these *.htaccess* files in their home directories. The user-defined directives can sometimes add insecure options to the server by overriding directives in the server configuration file. Directives can be placed in the server configuration file to prevent the server from using these files thereby preventing Administrative directives to be overridden and closing the potential security hole.

B. APACHE ADD-ON MODULES

Several of the Apache add-on modules are security related or can contribute to the overall security of the server. Directives for these modules placed in the server configuration file can create tighter security restrictions and access. Some of the more

common ones are discussed below. A complete listing of Apache provided modules is provided in Appendix I.

1. mod_access: This module provides access control to directories based on the client hostname or IP address that is sent in a request header field. Access to the server on the XTS-300 is currently designed to be through the Trusted Path. This access is controlled by user identification and authorization and the session level. This module could be used within a company to provide additional policy granularity to discriminate among departments. If each department had different subnet addresses, directories could be allowed or denied access to according to the department subnet addresses. This module is configured in this implementation using the default values in the server configuration file. It could be useful in a MLS LAN environment to provide additional policy granularity.

2. mod_auth, mod_auth_anon, mod_auth_db, mod_auth_dbm, mod_auth_digest, mod_digest: These modules provide for user authentication using various methods. With the NPS MLS LAN, the user is already identified and authorized before the server is started. These modules did not need to be configured in this implementation. It is recommended that some type of authentication method be used whether through Apache-provided modules or implementation provided.

3. mod_autoindex: This module provides an automated file listing of a directory on the server if one is not present. By default, this module looks for an *index.html* file in the directory requested by the client. If this file is not present, the server creates a directory listing of all files in the directory and sends this listing in response to the

client's request. This module could be very informational and provide a great amount of detail about a directory. It could be used in a covert channel. The client could be searching directories looking for files the server can access. It is recommended that this module not be configured in a MLS LAN environment.

4. mod_cern_meta: This module allows for CERN metafile semantics. Metafiles are HTTP headers that can be output in addition to the normal range of headers for each file accessed. This module can be used to provide a way to send extra information about a file, such as classification level, in a server response header line. This module is not configured in this implementation, but could be useful in a MLS LAN environment to provide additional information about the objects being served.

5. mod_cgi, mod_actions: These modules provide for execution of Common Gateway Interface (CGI) scripts. The mod_cgi module can be used with the mod_include module to support server side includes (SSI) that execute cgi-scripts. It has been well documented that allowing users this capability (i.e. cgi) can very risky and insecure. If users are allowed to write their own scripts they can introduce many security holes, allowing crackers to find out information about the server, or even delete and overwrite files [Ref. 28]. To overcome the security issues, the server should only provide trusted cgi-scripts and not allow users to create their own [Ref. 29]. There are several useful web sites that address security issues related to CGI including the World Wide Web Consortium web site [Ref. 30] and SecurityPortal web site [Ref. 31]. The mod_cgi module is configured in this implementation using the default values in the server configuration file. The mod_actions module is not configured

with this implementation. If these modules are configured in a MLS LAN environment, it is recommended that the Administrator fully research the security issues to ensure correct configuration.

6. mod_headers: This module provides for user customizable response headers. This can be used to provide information about the classification of the object, the session level of the user or other security related parameters. This module is not configured in this implementation but could be useful in a MLS LAN environment to provide additional security related information about the objects the server is serving.

7. mod_include: This module provides a handler for parsing files that include references to other files within them. It allows the server to dynamically build or add to a web page before sending it to the client. Using the server side include (SSI) statements in the web page, the server constructs the document before it is sent to the client. This is one way the server can handle web pages that contain links to higher levels of information. If the client is at a low session level and requests a page that has objects that are at higher levels, the high level objects will not be found and not sent to the client. Many of the security issues related to CGI are also related to SSI. Another security concern is the client's possible knowledge of the existence of files at higher levels. This module is currently configured in this server implementation to show proof of concept on creating dynamic web pages that contain references to different levels of classification. The use of SSI demonstrated the enforcement of the security policies on the server. This module could be useful in a MLS LAN environment to provide dynamic web pages containing objects at varying classification levels.

8. mod_info: This module provides comprehensive information about the server configuration. It provides information about the server configuration file, modules built into the server, and log file names. This is a very informational module but clients should not have access to the information it provides. It is recommended that this module not be configured in a MLS LAN environment.

9. mod_log_config: This module provides a file logging mechanism. Several log files are supported including a request log that logs all requests made to the server. The request log file contains information such as the object being requested and the requesting client address. These log files, if used, need to be read-protected so clients can not read them. If users are allowed to read the customized error log, the information could be invaluable to a hacker as it can reveal problems with the server configuration and CGI scripts [Ref. 32]. However in this implementation, they need to be write-able by users to allow the server to write to them. They must also exist at all levels of classification. This module is not configured in this implementation. The default error log provided enough information for this implementation and a customized error log was not configured. Use of a ring mechanism might allow privileged access to these files while preventing casual access by users. This module could be useful in a MLS LAN environment to keep track of cgi-scripts and other objects that clients are requesting or to create custom logs for the server.

10. mod_setenvif: This module is used to set environment variables based on the attributes of the request. It is used with mod_access to control access based on information in the client's header lines, i.e. whether the request has been redirected

from another untrusted site, information about the browser (secure or insecure). This module could be used within a company to provide additional policy granularity to discriminate among departments. This module is not configured in this implementation but could be useful in a MLS LAN environment to provide additional policy granularity.

11. mod_speling: This module attempts to correct misspellings of URLs. This module is also very large and will greatly increase the size of the server. This is not a major security concern but clients knowing that this module is present, could try guessing games to see if they can access files on the server that they would otherwise not know were there. This module is not configured in this implementation and is not recommended in a MLS LAN environment.

12. mod_status: This module provides information on the server status, activity, and performance. The client should have no need to know this information. With the ServerType as inetd, this module is useless. This module is not needed in this implementation.

13. mod_userdir: This module allows users to have home web pages on the server. This privilege could lead to security issues if the user has cgi-scripts that others can access or if the cgi-script is badly written. This module is configured using the default values in the server configuration file. If this module is configured in a MLS LAN environment, the security issues related to the module should be investigated.

14. mod_usertrack: This module allows the server to use cookies to track user activity. The server generates a "cookie", a unique identification number, and sends this

cookie to the client who stores the cookie in a file located on its machine. The only thing the client needs to store is the cookie number and the web server it was received from. The server stores information about the user (name, credit card number, preferences) in a log file on the server machine using the cookie as in index and key. Each time the client/user makes a request to this server, it will include the cookie. The server will use the cookie as an index into the file to retrieve information specific to the user. This cookie log file should not be readable by clients because the information contained in the file could be used by hackers to pose as other users to gain access to their information. This module is not configured in this implementation. If this module is configured in a MLS LAN environment, the log file needs to be created with the correct user access.

15. mod_vhost_alias: This module creates dynamically configured virtual hosts (see Chapter II-A for an explanation on virtual hosts). This module, like mod-access, could be used within a company to provide additional policy granularity to discriminate among departments. This module is not configured in this implementation but could useful in a MLS LAN environment to provide addition policy granularity.

C. SECURE SOCKET LAYER

Secure Socket Layer (SSL), also known as Transport Layer Security (TLS), is an encrypted communications protocol used to send information securely across networks. It is a layer that sits between the web server and the TCP/IP layers, transparently handling decryption and encryption on secure connections [Ref. 15]. The IETF RFC 2817 discusses using SSL/TLS with HTTP [Ref. 33]. SSL uses the HTTPS protocol ID and

listens to port number 443. Incorporating SSL into an Apache-based web server is more involved than just having the server listen to port 443 for requests. The Apache Software Foundation has a project called Apache-SSL that outlines the steps for adding SSL into an Apache-based web server and is described at its web site at www.apache-ssl.org [Ref. 34]. This Apache project describes what needs to be done to make an Apache-based web server support SSL.

There is also a commercially available third party add-on module, mod_ssl, that can be used with an Apache-based server. It can be downloaded from the mod_ssl web site, www.modssl.org [Ref. 35], along with instructions for incorporating it into an Apache-based server. SSL libraries from OpenSSL [Ref. 36] are also needed to support both the Apache-SSL software and the mod_ssl software. These can be downloaded from the OpenSSL web site, www.openssl.org [Ref. 35]. The OpenSSL libraries must be configured and compiled on the same platform as the Apache-based server. Once configured and compiled, the Apache-SSL software and/or the mod_ssl software can be compiled. At one time, the United States also required the RSAREF library for SSL in order to comply with patent regulations in the United States [Ref. 37]. This patent regulation has since expired. The RSAREF libraries had to be configured and compiled before the mod_ssl module could be compiled. The mod_ssl module relies on functions defined in the OpenSSL libraries. The mod_ssl module actually patches the Apache web server source code, so mod_ssl can not be built into the Apache server like most of the other add-on modules. The process is rather detailed and a step-by-step procedure needs to be followed to correctly obtain an Apache-based server that can support SSL. After the

compilation, there are several SSL server directives that need to be configured in the server configuration file.

Once the Apache-based SSL server has been compiled and configured to listen to port 443, the job is not complete. The server should have a private key and a signed certificate from a recognized authority (IETF RFC 2585 discusses public keys and HTTP [Ref. 38]). These procedures described above are the minimum required to configure SSL into an Apache-based server. More steps are required if advanced SSL options are being considered. Incorporating SSL into an Apache-based web server takes time, research, software resources other than the Apache software, and testing. This undertaking is beyond the scope of this thesis and is referred as future follow-on work in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

This chapter provides answers to the questions asked in the first chapter. It suggests some topics for future work related to this thesis.

A. DISCUSSION

Several questions were posed at the beginning of this thesis questioning the possibility and ease of implementation of an Apache-based server on a high assurance multilevel secure platform and the effects it would have on the trustworthiness of this platform. All of these questions were answered indirectly throughout the thesis. The overall result is that an Apache-based server was successfully modified to operate on a high assurance host acting as a server in a MLS LAN.

The structure of the Apache software is designed to support customized ports and platform specific web servers. The majority of the modifications was limited to a select few files. These were: two configuration scripts, one C language header file, and two C language source code files. These centralized changes will allow future Apache upgrades to be accomplished with relative ease on the XTS-300.

This Apache-based server is constrained by access controls and policies enforced by the XTS-300 Trusted Computing Base. It causes no adverse side affects to this environment. The Apache-based server is an application layer protocol server that the MLS LAN Secure Session Server controls. It should not affect the trustworthiness or evaluation rating of the XTS-300.

B. FUTURE WORK

This thesis can provide many follow-on projects.

The Apache Software Foundation released a new version, 1.3.14, of the Apache web server just before this thesis was complete. There is also the 2.0 version that requires the GNU tools autoconf and libtool. Follow-on work could include downloading these versions and incorporating the changes made for the XTS Apache-based web server. This will provide the latest version of an Apache-based web server and also the GNU tools.

Another follow on thesis would be to incorporate Secure Socket Layer (SSL) into the server. This was briefly discussed in Chapter V. Incorporating and testing an SSL Apache-based server would take considerable time and effort. This effort would provide an Apache-based server capable of serving *https* requests. It would also provide a server that is up to date with current Internet standards and advances. Additional study would be needed to determine if there are any security advantages to adding SSL for finer grained policies at the application level.

Follow-on and continued work would be learning and investigating the many capabilities of an Apache-based web server. The Apache server has many add-on modules that add great flexibility and capabilities to servers. These were not explored in this thesis but should be explored in the future to maximize the utility of the server.

A final area of interest would be to continue experiments with server side includes (SSI) to create dynamic web pages. The use of SSIs was briefly investigated in this research to show proof of concept.

C. CONCLUSIONS

This implementation of an Apache-based server on a high assurance multilevel platform was a success. The implementation process came with positive and negative experiences. The Apache-provided code was structured and written in such a way that modifications could be easily made and incorporated. This was a positive experience because it allowed the modifications to be centralized in a small amount of code. Working on the XTS-300 platform without a good debugging tool was difficult and challenging at times. This was a negative experience that added complications to the testing of the Apache-based server. However, it also provided a positive experience because without these complications the underlying timing issue might not have been noticed and corrected. The overall experience from this thesis was a very positive one, mostly due to the talented team of professionals that supported and helped with the effort.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: GLOSSARY

Apache	A commercially available HTTP web server application written by the Apache Software Foundation Project.
CGI	Common Gateway Interface. A protocol for scripts to gather information from a user request and respond to it.
Client	An application program that establishes connections for the purpose of sending requests.
COTS	Commercial-off-the-shelf
CR	Carriage return
DAC	Discretionary Access Control. A means of restricting access to objects based on the identity of subjects and/or groups to which they belong [Ref. 19].
DoD	Department of Defense
FTP	File Transfer Protocol. The standard protocol of the Internet architecture for transferring files between hosts [Ref. 39].
GIF	Graphical Interchange Format.
GNU	A foundation that provides software products and tools.
HTML	HyperText Markup Language. A language used to construct World Wide Web pages [Ref. 39].
HTTP	HyperText Transfer Protocol. An application level protocol based on a request/reply paradigm and used in the World Wide Web [Ref. 39].
IETF	Internet Engineering Task Force. A task force responsible for providing short-term engineering solutions for the Internet [Ref. 39].
Internet	The global internet based on the Internet (TCP/IP) architecture, connecting millions of hosts worldwide [Ref. 39].
IP	Internet Protocol. A protocol that provides a connectionless, best-effort delivery service of datagrams across the Internet [Ref. 39].
JPEG	An image in the Joint Photographic Experts Group format [Ref. 39].
LAN	Local Area Network. A network based on any physical network technology that is designed to span distances of up to a few thousand meters [Ref. 39].
LF	Line Feed
MAC	Mandatory Access Control. A means of restricting access to objects based on the sensitivity of the information contained in the objects and the

	formal authorization of subjects to access information of such sensitivity [Ref. 19].
MLS	Multilevel Security. The ability of a system to contain information with different sensitivities that simultaneously permits access by users with different security clearances and needs-to-know, but prevents users from obtaining access to information for which they lack authorization [Ref. 40].
NPS	Naval Postgraduate School
RFC	Request For Comment
SAK	Secure Attention Key
Server	The provider of a service in a client/server distributed system [Ref. 39].
SMTP	Simple Mail Transfer Protocol
SSS	Secure Session Server
SSI	Server Side Include
SSL	Secure Socket Layer. A protocol layer that runs over TCP to provide authentication and encryption of connections. Also known as Transport Layer Security (TLS) [Ref. 39].
STOP	The Unix-like Operating System used on the Wang XTS-300 platform.
TCB	Trusted Computing Base
TCBE	Trusted Computing Base Extension
TCP/IP	Transmission Control Protocol over Internet Protocol
TCP	Transmission Control Protocol. Connection-oriented transport protocol of the Internet architecture. TCP provides a reliable, byte-stream delivery service [Ref. 39].
TCSEC	Trusted Computer System Evaluation Criteria
Telnet	Remote terminal protocol of the Internet architecture. Telnet allows you to interact with a remote system as if your terminal is directly connected to that machine [Ref. 39].
TLS	Transport Layer Security. Also known as Secure Socket Layer.
TPS	Trusted Path Server.
URI	Uniform resource identifier. Formatted strings that identify, via name, location, or any other characteristic, a resource. [Ref. 5] (Defined in IETF RFC 2396 [Ref. 6].)
URL	Uniform resource locator. Formatted strings that identify the location of an object. (Defined in IETF RFC 2396 [Ref. 6].)

WWW	World Wide Web. A hypermedia information service on the Internet [Ref. 39].
XTS-300	A combination of a multilevel secure operating system and a Pentium based computer.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: APACHE SOFTWARE LICENSE FILE

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: DIRECTORY LISTING

This is a complete directory listing of the Apache-based server files on the XTS-300. The top-level directory is *apache13*. Files that were modified for this implementation are in bold. Files in bold-italics only had debug statements added to various functions with no changes to the Apache-provided source code.

```
apache13/:
ABOUT_APACHE        WARNING-NT.TXT      configure           license
Announcement        cgi-bin/            htdocs/             readme
Makefile            conf/               icons/              readme.nt
Makefile.tmpl       config.layout       install             src/
README.configure    config.status*      keys

apache13/cgi-bin:
printenv    test-cgi

apache13/conf:
access.conf-dist        httpd.conf-dist-win     srm.conf-dist
access.conf-dist-win    httpd.conf-dist.nw      srm.conf-dist-win
highperf.conf-dist      magic
httpd.conf-dist         mime.types

apache13/htdocs:
apache_pb.gif           index.html.en           index.html.nl
index.html.ca           index.html.es           index.html.po.iso-pl
index.html.cz           index.html.fr           index.html.pt
index.html.de           index.html.it           index.html.pt-br
index.html.dk           index.html.ja.jis       index.html.se
index.html.ee           index.html.lu           manual/

apache13/htdocs/manual:
bind.html               install.html            process-model.html
cgi_path.html           invoking.html           readme-tpf.html
content-nego.html       keepalive.html          search/
custom-error.html       license                 sections.html
dns-caveats.html        location.html           sourcereorg.html
dso.html                man-template.html       stopping.html
ebcdic.html             misc/                   suexec.html
env.html                mod/                    suexec_1_2.html
footer.html             multilogs.html          unixware.html
handler.html            netware.html            upgrading_to_1_3.html
header.html             new_features_1_0.html   vhosts/
images/                 new_features_1_1.html   windows.html
index.html              new_features_1_2.html
install-tpf.html        new_features_1_3.html

apache13/htdocs/manual/images:
custom_errordocs.gif    mod_rewrite_fig1.fig    mod_rewrite_fig2.gif
home.gif                mod_rewrite_fig1.gif    sub.gif
index.gif               mod_rewrite_fig2.fig

apache13/htdocs/manual/misc:
API.html                footer.html             perf-hp.html
FAQ.html                header.html             perf-tuning.html
HTTP_Features.tsv       howto.html              perf.html
client_block_api.html   index.html              rewriteguide.html
compat_notes.html       known_problems.html     security_tips.html
```

custom_errordocs.html nopgp.html vif-info.html
descriptors.html perf-bsd44.html windoz_keepalive.html
fin_wait_2.html perf-dec.html

apache13/htdocs/manual/mod:

core.html	mod_cern_meta.html	mod_log_config.html
directive-dict.html	mod_cgi.html	mod_log_referer.html
directives.html	mod_cookies.html	**mod_mime.html**
footer.html	mod_digest.html	mod_mime_magic.html
header.html	mod_dir.html	mod_mmap_static.html
index.html	mod_dld.html	**mod_negotiation.html**
mod_access.html	mod_dll.html	mod_proxy.html
mod_actions.html	mod_env.html	mod_rewrite.html
mod_alias.html	mod_example.html	mod_setenvif.html
mod_asis.html	mod_expires.html	mod_so.html
mod_auth.html	mod_headers.html	mod_speling.html
mod_auth_anon.html	mod_imap.html	mod_status.html
mod_auth_db.html	mod_include.html	mod_unique_id.html
mod_auth_dbm.html	mod_info.html	mod_userdir.html
mod_auth_digest.html	mod_isapi.html	mod_usertrack.html
mod_autoindex.html	mod_log_agent.html	mod_vhost_alias.html
mod_browser.html	mod_log_common.html	

apache13/htdocs/manual/search:
manual-index.cgi

apache13/htdocs/manual/vhosts:

details.html	header.html	name-based.html
details_1_2.html	host.html	vhosts-in-depth.html
examples.html	index.html	virtual-host.html
fd-limits.html	ip-based.html	
footer.html	mass.html	

apache13/icons:

a.gif	dir.gif	link.gif	screw2.gif
alert.black.gif	down.gif	movie.gif	script.gif
alert.red.gif	dvi.gif	p.gif	small/
apache_pb.gif	f.gif	patch.gif	sound1.gif
back.gif	folder.gif	pdf.gif	sound2.gif
ball.gray.gif	folder.open.gif	pie0.gif	sphere1.gif
ball.red.gif	folder.sec.gif	pie1.gif	sphere2.gif
binary.gif	forward.gif	pie2.gif	tar.gif
binhex.gif	generic.gif	pie3.gif	tex.gif
blank.gif	generic.red.gif	pie4.gif	text.gif
bomb.gif	generic.sec.gif	pie5.gif	transfer.gif
box1.gif	hand.right.gif	pie6.gif	unknown.gif
box2.gif	hand.up.gif	pie7.gif	up.gif
broken.gif	icon.sheet.gif	pie8.gif	uu.gif
burst.gif	image1.gif	portal.gif	uuencoded.gif
c.gif	image2.gif	ps.gif	world1.gif
comp.blue.gif	image3.gif	quill.gif	world2.gif
comp.gray.gif	index.gif	readme	
compressed.gif	layout.gif	right.gif	
continued.gif	left.gif	screw1.gif	

apache13/icons/small:

README.txt	compressed.gif	image.gif	sound2.gif
back.gif	continued.gif	image2.gif	tar.gif
binary.gif	dir.gif	index.gif	text.gif
binhex.gif	dir2.gif	key.gif	transfer.gif
blank.gif	doc.gif	movie.gif	unknown.gif
broken.gif	forward.gif	patch.gif	uu.gif
burst.gif	generic.gif	ps.gif	
comp1.gif	generic2.gif	rainbow.gif	
comp2.gif	generic3.gif	sound.gif	

```
apache13/src:
Apache.dsp              Configure.orig          httpd.good*
Apache.mak              Makefile                include/
ApacheCore.def          Makefile.config         install
ApacheCore.dsp          Makefile.nt             junk
ApacheCore.mak          Makefile.tmpl           lib/
ApacheCoreOS2.def       Makefile_win32.txt      main/
ApacheNW.mcp            README.EBCDIC           modules/
ApacheNW.mcp.gz         ap/                     modules.c
BUILD.NOTES             apachenw.mcp            modules.c.old
Configuration           apaci*                  modules.o
Configuration.1024      buildmark.c             os/
Configuration.new       buildmark.o             porting
Configuration.tmpl      changes                 readme
Configure               helpers/                regex/
Configure.1026          httpd*                  support/

apache13/src/ap:
Makefile            ap_checkpass.o      ap_getpass.c        ap_signal.c
Makefile.tmpl       ap_cpystrn.c        ap_getpass.c.orig   ap_signal.o
ap.dsp              ap_cpystrn.o        ap_getpass.o        ap_slack.c
ap.mak              ap_execve.c         ap_md5c.c           ap_slack.o
ap_base64.c         ap_execve.o         ap_md5c.o           ap_snprintf.c
ap_base64.o         ap_fnmatch.c        ap_sha1.c           ap_snprintf.o
ap_checkpass.c      ap_fnmatch.o        ap_sha1.o           libap.a

apache13/src/helpers:
CutRule          PrintPath        find-dbm-lib     mfhead
GuessCodeset     TestCompile      findcpp.sh       mkdir.sh
GuessOS          binbuild.sh      fmn.sh           mkshadow.sh
GuessOS.orig     buildinfo.sh     fp2rp            ppl.sh
MakeEtags        checkheader.sh   getuid.sh        slo.sh
MakeLint         dummy.c          install.sh

apache13/src/include:
alloc.h                 compat.h                http_vhost.h
ap.h                    conf.h                  httpd.h
ap_compat.h             explain.h               httpd.h.orig
ap_config.h             fnmatch.h               multithread.h
ap_config.h.bu          hsregex.h               rfc1413.h
ap_config.h.orig        http_conf_globals.h     scoreboard.h
ap_config_auto.h        http_config.h           util_date.h
ap_ctype.h              http_core.h             util_md5.h
ap_md5.h                http_log.h              util_script.h
ap_mmn.h                http_main.h             util_uri.h
ap_sha1.h               http_protocol.h
buff.h                  http_request.h

apache13/src/lib:
Makefile     expat-lite/

apache13/src/lib/expat-lite:
Makefile        hashtable.h     xmldef.h        xmlrole.c       xmltok.mak
Makefile.tmpl   hashtable.o     xmlparse.c      xmlrole.h       xmltok.o
asciitab.h      iasciitab.h     xmlparse.def    xmlrole.o       xmltok_impl.c
changes         latin1tab.h     xmlparse.dsp    xmltok.c        xmltok_impl.h
dllmain.c       libexpat.a      xmlparse.h      xmltok.def      xmltok_ns.c
expat.html      nametab.h       xmlparse.mak    xmltok.dsp
hashtable.c     utf8tab.h       xmlparse.o      xmltok.h

apache13/src/main:
Makefile                http_config.c           http_vhost.c
Makefile.tmpl           http_config.c.orig      http_vhost.o
alloc.c                 http_config.o           libmain.a
```

```
alloc.c.orig            http_core.c             rfc1413.c
alloc.o                 http_core.c.orig        rfc1413.o
buff.c                  http_core.o             test_char.h
buff.c.orig             http_log.c              uri_delims.h
buff.o                  http_log.c.orig         util.c
gen_test_char*          http_log.o              util.c.orig
gen_test_char.c         http_main.c             util.o
gen_test_char.dsp       http_main.c.orig        util_date.c
gen_test_char.mak       http_main.o             util_date.o
gen_test_char.o         http_protocol.c         util_md5.c
gen_uri_delims*         http_protocol.c.orig    util_md5.o
gen_uri_delims.c        http_protocol.o         util_script.c
gen_uri_delims.dsp      http_request.c          util_script.o
gen_uri_delims.mak      http_request.c.orig     util_uri.c
gen_uri_delims.o        http_request.o          util_uri.o

apache13/src/modules:
Makefile        experimental/  modules__       readme
example/        extra/         proxy/          standard/

apache13/src/modules/example:
Makefile.tmpl   mod_example.c   readme

apache13/src/modules/experimental:
Makefile.tmpl       mod_auth_digest.c   mod_mmap_static.c

apache13/src/modules/extra:
Makefile.tmpl

apache13/src/modules/proxy:
ApacheModuleProxy.dsp   Makefile.tmpl           proxy_connect.c
ApacheModuleProxy.mak   mod_proxy.c             proxy_ftp.c
Makefile.OS2            mod_proxy.h             proxy_http.c
Makefile.libdir         proxy_cache.c           proxy_util.c

apache13/src/modules/standard:
Makefile            mod_cern_meta.c     mod_mime.c
Makefile.OS2        mod_cgi.c           mod_mime.o
Makefile.tmpl       mod_cgi.o           mod_mime_magic.c
libstandard.a       mod_digest.c        mod_negotiation.c
mod_access.c        mod_dir.c           mod_negotiation.o
mod_access.o        mod_dir.o           mod_rewrite.c
mod_actions.c       mod_env.c           mod_rewrite.h
mod_alias.c         mod_env.o           mod_setenvif.c
mod_alias.o         mod_expires.c       mod_so.c
mod_asis.c          mod_headers.c       mod_speling.c
mod_asis.o          mod_imap.c          mod_status.c
mod_auth.c          mod_imap.o          mod_status.o
mod_auth.o          mod_include.c       mod_unique_id.c
mod_auth_anon.c     mod_include.o       mod_userdir.c
mod_auth_db.c       mod_info.c          mod_userdir.o
mod_auth_db.module  mod_info.o          mod_usertrack.c
mod_auth_dbm.c      mod_log_agent.c     mod_vhost_alias.c
mod_autoindex.c     mod_log_config.c
mod_autoindex.o     mod_log_referer.c

apache13/src/os:
bs2000/   mpeix/    netware/  os2/      os390/    tpf/      unix/     win32/

apache13/src/os/bs2000:
Makefile.tmpl   ebcdic.c        os-inline.c     os.h
bs2login.c      ebcdic.h        os.c

apache13/src/os/mpeix:
Makefile.tmpl    gettimeofday.c   os-inline.c      os.h
```

```
dlopen.c          mpe_dl_stub.c     os.c              readme

apache13/src/os/netware:
Apache.def                ApacheModuleRewrite.def   multithread.c
ApacheCore.imp            ApacheModuleSpeling.def   os.c
ApacheCoreNW.def          ApacheModuleStatus.def    os.h
ApacheModuleDigest.def    getopt.c                  precomp.h
ApacheModuleExpires.def   getopt.h                  test_char.h
ApacheModuleHeaders.def   main_nlm.c                uri_delims.h
ApacheModuleInfo.def      mod_nlm.c
ApacheModuleProxy.def     modules.c

apache13/src/os/os2:
Makefile.tmpl   os-inline.c     os.c            os.h            util_os2.c

apache13/src/os/os390:
Makefile.tmpl   ebcdic.c        os-inline.c     os.h
README.os390    ebcdic.h        os.c            xebcdic.sh

apache13/src/os/tpf:
Makefile.tmpl   cgetop.c        ebcdic.h        os.c            samples/
TPFExport       ebcdic.c        os-inline.c     os.h

apache13/src/os/tpf/samples:
linkdll.jcl   loadset.jcl

apache13/src/os/unix:
Makefile        libos.a         os-inline.c     os.c            os.o
Makefile.tmpl   os-aix-dso.c    os-inline.o     os.h

apache13/src/os/win32:
ApacheModuleDigest.dsp    ApacheOS.dsp              multithread.c
ApacheModuleDigest.mak    ApacheOS.mak              os.c
ApacheModuleExpires.dsp   MakeModuleMak.cpp         os.h
ApacheModuleExpires.mak   MakeModuleMak.mak         passwd.c
ApacheModuleHeaders.dsp   Module.mak.tmpl           passwd.h
ApacheModuleHeaders.mak   apache.ico                readdir.c
ApacheModuleInfo.dsp      apache.rc                 readdir.h
ApacheModuleInfo.mak      getopt.c                  registry.c
ApacheModuleRewrite.dsp   getopt.h                  registry.h
ApacheModuleRewrite.mak   installer/                resource.h
ApacheModuleSpeling.dsp   main_win32.c              service.c
ApacheModuleSpeling.mak   mod_dll.c                 service.h
ApacheModuleStatus.dsp    mod_isapi.c               util_win32.c
ApacheModuleStatus.mak    modules.c

apache13/src/os/win32/installer:
apache.iwz   installdll/  readme.txt

apache13/src/os/win32/installer/installdll:
core          install.def   install.mak
install.c     install.dsp   test/

apache13/src/os/win32/installer/installdll/test:
resource.h   test.def     test.h       test.mak
test.c       test.dsp     test.ico     test.rc

apache13/src/regex:
COPYRIGHT       engine.ih       regcomp.o       regex.mak       tests
Makefile        libregex.a      regerror.c      regex2.h        utils.h
Makefile.tmpl   main.c          regerror.ih     regexec.c       whatsnew
cclass.h        mkh             regerror.o      regexec.o
cname.h         readme          regex.3         regfree.c
debug.c         regcomp.c       regex.7         regfree.o
engine.c        regcomp.ih      regex.dsp       split.c
```

```
apache13/src/support:
Makefile            htdigest.c          htpasswd.mcp.gz     rotatelogs*
Makefile.tmpl       htdigest.dsp        htpasswd.o          rotatelogs.8
ab.8                htdigest.mak        httpd.8             rotatelogs.c
ab.c                htdigest.mcp        httpd.exp           rotatelogs.o
apachectl           htdigest.mcp.gz     log_server_status   sha1/
apachectl.8         htdigest.o          logresolve*         split-logfile
apxs.8              htpasswd*           logresolve.8        suexec.8
apxs.pl             htpasswd.1          logresolve.c        suexec.c
dbmmanage           htpasswd.c          logresolve.o        suexec.h
dbmmanage.1         htpasswd.dsp        logresolve.pl
htdigest*           htpasswd.mak        phf_abuse_log.cgi
htdigest.1          htpasswd.mcp        readme

apache13/src/support/sha1:
README.sha1         convert-sha1.pl     htpasswd-sha1.pl    ldif-sha1.example
```

APPENDIX D: MODIFICATIONS TO CONFIGURATION FILES

The following additions were made to the *GuessOS* file:

```
##############################
#
# ADDED FOR XTS-300
#
##############################
      *:stop4.*:STOP:*)
       echo "${SYSTEM}-xts300-stop4.4"; exit 0
       ;;
##############################
```

The following additions were made to the *Configure* file:

```
#########################
#
# ADDED FOR XTS-300
#
#########################
   *-xts300-stop4.*)
       OS='XTS'
       CFLAGS="$CFLAGS    -g    -DXTS    -I/usr2/shifflet/wip/include    -
DUSE_P_SOCKET"
       LDFLAGS="$LDFLAGS -lcass -lmw -lsocket -lgen"
       LIBS="$LIBS  -L/lib  -lsocket  -lmw  -lgen  -L/usr/lib  -lcrypt  -
L/usr2/shifflet/wip/lib -lut_cass -lcass"
       RANLIB=true
       LN=ln
       ;;
########################
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E: MODIFICATIONS TO SOURCE CODE

All changes and additions are indicated by bold face type.

1. The following changes were made to *ap_getpass()* function in the *ap_getpass.c* file:

```
#if defined (XTS) /* added for XTS */
     (int) pw_got = getpass(prompt);
#else
      pw_got = getpass(prompt);
#endif       /* end add for XTS */
```

2. The following line was modified in the *initgroup()* function in the *util.c* file:

```
#if defined(QNX) || defined(MPE) || defined(BEOS) || defined(TPF) ||
defined(__TANDEM) || defined(NETWARE) || defined(XTS) /* mod for XTS */
```

3. The following two functions, *ap_read()* and *ap_write()*, were found to be the low-level functions that called the XTS system *read()* and *write()* functions. They are defined in the *buff.c* source file. The changes made allow the functions to communicate with both pseudo socket and network sockets.

```
/* the lowest level reading primitive */
static int ap_read(BUFF *fb, void *buf, int nbyte)
{

#ifdef USE_P_SOCKET      /* added for XTS */
extern int my_fd;
#endif                   /* end add for XTS */

    int rv;

#ifdef WIN32
    if (fb->hFH != INVALID_HANDLE_VALUE) {
        if (!ReadFile(fb->hFH,buf,nbyte,&rv,NULL)) {
            errno = GetLastError();
            rv = -1;
        }
```

```
    }
    else
#endif
      rv = PREAD(fb->fd_in, buf, nbyte); /* mod for XTS */

    return rv;
}


/* the lowest level writing primitive */
static int ap_write(BUFF *fb, const void *buf, int nbyte)
{

#ifdef USE_P_SOCKET      /* added for XTS */
extern int my_fd;
#endif                   /* end add for XTS */

    int rv;

#ifdef WIN32
    if (fb->hFH != INVALID_HANDLE_VALUE) {
        if (!WriteFile(fb->hFH,buf,nbyte,&rv,NULL)) {
            errno = GetLastError();
            rv = -1;
        }
    }
    else
#endif
#if defined (B_SFIO)
      rv = sfwrite(fb->sf_out, buf, nbyte);
#else
#ifdef _OSD_POSIX
/* Sorry, but this is a hack: On BS2000, currently the send() call
* has slightly better performance, and it doesn't have a maximum
* transfer size of 16kB per write. Both write() and writev()
* currently have such a limit and therefore don't work
* too well with MMAP files.
*/
      if (fb->flags & B_SOCKET)
          rv = send(fb->fd, buf, nbyte, 0);
      else
#endif
TMP_OUT_1(tmpf_outbuff, "ap_write nbyte [%d] \n", nbyte, tmpf);

      rv = PWRITE(fb->fd, buf, nbyte); /* mod for XTS */
#endif

    return rv;
}
```

4. This following lines of code were added to the beginning of the source code file *http_main.c*. They define a signal handler and the functions needed to communicate with the pseudo sockets. It also sets up parameters for the debug file.

```
/*************************
*
* ADDED FOR XTS
*
*************************/
#ifdef USE_P_SOCKET
   int my_fd;
   char tmpf_out[80];
   FILE *tmpf = NULL;

   // internal functions to handle read with timeout
   void handler(int signo)
   {
     // do nothing, just return
     int debug_on = 0;
if (tmpf) {
   TMP_OUT_1(tmpf_out,"received signal [%d]\n", signo, tmpf);
}
     dbugd(debug_on, "handler(): entered signo = " , signo);
   }


   /* Wait for input to be available in the PSKT
    * Accepts: timeout in seconds
    * Returns: 1 if have input, else 0
    */

   long server_input_wait (long seconds)
   {
     long result;
     int sel_res;
     fd_set rfd, xfd;
     struct timeval tmo;
     tmo.tv_sec = seconds; tmo.tv_usec = 0;
     FD_ZERO (&rfd);
     FD_ZERO (&xfd);
     FD_SET (my_fd,&rfd);
     FD_SET (my_fd,&xfd);
     sel_res = pskt_select_cli(my_fd+1,&rfd,0,&xfd,&tmo);
     if (FD_ISSET(my_fd,&xfd))
     {
        result = -1;
     }else if (FD_ISSET(my_fd,&rfd))
     {
        result = sel_res ? 1 : 0;
     }else{
        result = 0;
     }
     return result;
   }

   int psin_with_pause(char *s, int length)
   {
      // Do a 'zero' delay select,
      // just in case data is waiting there
      // and we have already gotten the signal
      // from the SSS child process
// TMP_OUT_0("in psin_with_pause()\n",tmpf);
      long wait_res = server_input_wait(0);
```

```
      if (!wait_res)
      {
     // Nothing waiting, pause until data is available
      // TMP_OUT_0("in psin_with_pause() pausing\n",tmpf);
         pause();
      }
      if (wait_res == -1)
      {
// TMP_OUT_0("in psin_with_pause() returning 0\n",tmpf);
         return(0);
      }else{
         if (errno == EINTR) errno = 0;
// TMP_OUT_0("in psin_with_pause() calling
pskt_read_stop_at_cli()\n",tmpf);
         int tmp_res = pskt_read_from_cli(my_fd, s, length);
// TMP_OUT_0("in psin_with_pause() pskt_read_stop_at_cli() all
done\n",tmpf);
   TMP_OUT_1(tmpf_out,"Read len [%d]\n", tmp_res, tmpf);
         return(tmp_res);
      }
   }
   int pbin_with_pause()
   {
      // Do a 'zero' delay select,
      // just in case data is waiting there
      // and we have already gotten the signal
      // from the SSS child process
      long wait_res = server_input_wait(0);

      if (!wait_res)
      {
         // Nothing waiting, pause until data is available
         pause();
      }
      if (wait_res == -1)
      {
         return(0);
      }else{
         if (errno == EINTR) errno = 0;
         return(pskt_read_char_cli(my_fd));
      }
   }
   int pbout_handler(char c)
   {
      int again = 2;   // Retry twice after full buffer
      int result = 0;
      while ((result != EOF) && (result != 1)) {
         result = pskt_write_char_svr (my_fd, c);
         if (!result) {
            // The PSKT buffer must be full,
            if (!again) {
               errno = ENOSPC;
               result = EOF;
            }else{
               // Wait, then try again
               poll(NULL, 0, 100);  // Use poll to get 0.1 second delay
               again--;     // decrease remaining chances
            }
         }
      }
```

```
      return result;
   }
   int psout_handler(char *s, int total_len)
   {
      int again = 2;    // Retry twice after full buffer
      int result = 0;
      int tmplen = total_len;
      char *tmp_s = s;
TMP_OUT_1(tmpf_out,"psout_handler writing [%d]\n", tmplen, tmpf);
      while ((result != EOF) && (result != total_len)) {
         result = pskt_write_to_svr(my_fd, tmp_s, tmplen);
TMP_OUT_1(tmpf_out,"result is [%d]\n", result, tmpf);
         if (!result) {
            if (!again) {
               errno = ENOSPC;
               result = EOF;
            }else{
               poll(NULL, 0, 100);  // Use poll to get 0.1 second delay
               again--;      // decrease remaining chances
            }
         }else if (result != tmplen) {
            tmplen -= result;
            tmp_s += result;
            result = 0;
            again = 2;    // reset remaining chances
            poll(NULL, 0, 100);  // Use poll to get 0.1 second delay
         }else{
            // result = strlen(s);    // We wrote it all out, return full
size
            result = total_len;    // We wrote it all out, return full
size
         }
      }
      return result;
   }

#endif
```

5. The following lines were added to the *http_main.c* source file to setup the SIGURG signal handler:

```
/*************************
*
* ADDED FOR XTS
*
*************************/

#ifdef SIGURG
    signal(SIGURG, timeout);
#endif
```

6. The following lines were changed in *http_main.c* because the XTS-300 *fcntl()* function expects different data types than those being used:

```
while ((ret = fcntl(lock_fd, F_SETLKW, (int)&lock_it)) < 0 && errno ==
EINTR) { /* mod for XTS */
while ((ret = fcntl(lock_fd, F_SETLKW, (int)&unlock_it)) < 0 && errno ==
EINTR) { /* mod for XTS */
```

7. This is the main function call in *http_main.c*. The changes added for this Apache-based implementation are in bold.

```
int REALMAIN(int argc, char *argv[])
{
    int c;
    int sock_in;
    int sock_out;
    char *s;

/*************************
*
* ADDED FOR XTS
*
*************************/

#ifdef USE_P_SOCKET     /* added for XTS */

   char tmplogfile[80];
   sprintf(tmplogfile, "/tmp/http_log.tmp");
   tmpf = fopen(tmplogfile, "a+");
   if(tmpf == NULL) {
      printf("failed to open %s\n", tmplogfile);
   }
   if(chmod(tmplogfile, 438)) {
      printf("failed to chmod on %s\n", tmplogfile);
   }

//   FILE *tmpf2 = freopen(tmplogfile, "a", stdout);

   TMP_OUT_0("hello world\n",tmpf);
   TMP_OUT_1(tmpf_out,"Started HTTPD pid is [%d]\n", getpid(), tmpf);
#endif      // end add for XTS

#ifdef SecureWare
    if (set_auth_parameters(argc, argv) < 0)
      perror("set_auth_parameters");
    if (getluid() < 0)
      if (setluid(getuid()) < 0)
          perror("setluid");
    if (setreuid(0, 0) < 0)
      perror("setreuid");
#endif
```

```
#ifdef SOCKS
    SOCKSinit(argv[0]);
#endif

#ifdef TPF
    EBW_AREA input_parms;
    ecbptr()->ebrout = PRIMECRAS;
    input_parms = * (EBW_AREA *)(&(ecbptr()->ebw000));
#endif

    MONCONTROL(0);

    common_init();

    if ((s = strrchr(argv[0], PATHSEPARATOR)) != NULL) {
      ap_server_argv0 = ++s;
    }
    else {
      ap_server_argv0 = argv[0];
    }

    ap_cpystrn(ap_server_root, HTTPD_ROOT, sizeof(ap_server_root));
    ap_cpystrn(ap_server_confname, SERVER_CONFIG_FILE,
sizeof(ap_server_confname));

    ap_setup_prelinked_modules();

    while ((c = getopt(argc, argv,
                       "D:C:c:xXd:f:vVlLR:StTh"
#ifdef DEBUG_SIGSTOP
                       "Z:"
#endif
                )) != -1) {
      char **new;
      switch (c) {
      case 'c':
          new = (char **)ap_push_array(ap_server_post_read_config);
          *new = ap_pstrdup(pcommands, optarg);
          break;
      case 'C':
          new = (char **)ap_push_array(ap_server_pre_read_config);
          *new = ap_pstrdup(pcommands, optarg);
          break;
      case 'D':
          new = (char **)ap_push_array(ap_server_config_defines);
          *new = ap_pstrdup(pcommands, optarg);
          break;
      case 'd':
          ap_cpystrn(ap_server_root, optarg, sizeof(ap_server_root));
          break;
      case 'f':
          ap_cpystrn(ap_server_confname, optarg,
sizeof(ap_server_confname));
          break;
      case 'v':
          ap_set_version();
          printf("Server version: %s\n", ap_get_server_version());
          printf("Server built:   %s\n", ap_get_server_built());
          exit(0);
```

```
        case 'V':
            ap_set_version();
            show_compile_settings();
            exit(0);
        case 'l':
            ap_suexec_enabled = init_suexec();
            ap_show_modules();
            exit(0);
        case 'L':
            ap_show_directives();
            exit(0);
        case 'X':
            ++one_process;      /* Weird debugging mode. */
            break;
#ifdef TPF
        case 'x':
            os_tpf_child(&input_parms.child);
            set_signals();
            break;
#endif
#ifdef DEBUG_SIGSTOP
        case 'Z':
            raise_sigstop_flags = atoi(optarg);
            break;
#endif
#ifdef SHARED_CORE
        case 'R':
            /* just ignore this option here, because it has only
             * effect when SHARED_CORE is used and then it was
             * already handled in the Shared Core Bootstrap
             * program.
             */
            break;
#endif
        case 'S':
            ap_dump_settings = 1;
            break;
        case 't':
            ap_configtestonly = 1;
            ap_docrootcheck = 1;
            break;
        case 'T':
            ap_configtestonly = 1;
            ap_docrootcheck = 0;
            break;
        case 'h':
            usage(argv[0]);
        case '?':
            usage(argv[0]);
        }
    }

    ap_suexec_enabled = init_suexec();
    server_conf = ap_read_config(pconf, ptrans, ap_server_confname);


    if (ap_configtestonly) {
        fprintf(stderr, "Syntax OK\n");
        exit(0);
    }
```

```
    if (ap_dump_settings) {
        exit(0);
    }

    child_timeouts = !ap_standalone || one_process;

#ifdef BEOS
    /* make sure we're running in single_process mode - Yuck! */
    one_process = 1;
#endif

#ifndef TPF
    if (ap_standalone) {
      ap_open_logs(server_conf, plog);
      ap_set_version();
      ap_init_modules(pconf, server_conf);
      version_locked++;
      STANDALONE_MAIN(argc, argv);
    }
#else
    if (ap_standalone) {
        if(!tpf_child) {
            memcpy(tpf_server_name, input_parms.parent.servname,
                   INETD_SERVNAME_LENGTH);
            tpf_server_name[INETD_SERVNAME_LENGTH+1] = '\0';
            ap_open_logs(server_conf, pconf);
        }
        ap_set_version();
        ap_init_modules(pconf, server_conf);
        version_locked++;
        if(tpf_child) {
            copy_listeners(pconf);
            reset_tpf_listeners(&input_parms.child);
            server_conf->error_log = NULL;
#ifdef SCOREBOARD_FILE
            scoreboard_fd = input_parms.child.scoreboard_fd;
            ap_scoreboard_image = &_scoreboard_image;
#else /* must be USE_TPF_SCOREBOARD or USE_SHMGET_SCOREBOARD */
            ap_scoreboard_image =
                (scoreboard *)input_parms.child.scoreboard_heap;
#endif
            child_main(input_parms.child.slot);
        }
        else
            STANDALONE_MAIN(argc, argv);
    }
#endif
    else {
      conn_rec *conn;
      request_rec *r;
      struct sockaddr sa_server, sa_client;
      BUFF *cio;
      NET_SIZE_T l;

      ap_set_version();
      /* Yes this is called twice. */
      ap_init_modules(pconf, server_conf);
      version_locked++;
      ap_open_logs(server_conf, plog);
      ap_init_modules(pconf, server_conf);
```

```
      set_group_privs();

#ifdef MPE
      /* Only try to switch if we're running as MANAGER.SYS */
      if (geteuid() == 1 && ap_user_id > 1) {
          GETPRIVMODE();
          if (setuid(ap_user_id) == -1) {
            GETUSERMODE();
            ap_log_error(APLOG_MARK, APLOG_ALERT, server_conf,
                      "setuid: unable to change to uid: %d",
ap_user_id);
            exit(1);
          }
          GETUSERMODE();
      }
#else
      /* Only try to switch if we're running as root */
      if (!geteuid() && setuid(ap_user_id) == -1) {
          ap_log_error(APLOG_MARK, APLOG_ALERT, server_conf,
                  "setuid: unable to change to uid: %ld",
                  (long) ap_user_id);
          exit(1);
      }
#endif
      if (ap_setjmp(jmpbuffer)) {
          exit(0);
      }

#ifdef TPF
/* TPF's Internet Daemon passes the incoming socket nbr (inetd mode
only) */
    sock_in = sock_out = input_parms.parent.socket;
/* TPF also needs a signal set for alarm in inetd mode */
    signal(SIGALRM, alrm_handler);
#elif defined(MPE)
/* HP MPE 5.5 inetd only passes the incoming socket as stdin (fd 0),
whereas HPUX inetd passes the incoming socket as stdin (fd 0) and stdout
(fd 1).
   Go figure.  SR 5003355016 has been submitted to request that the
existing functionality be documented, and then to enhance the
functionality to be like HPUX. */

    sock_in = fileno(stdin);
    sock_out = fileno(stdin);
#else
    sock_in = fileno(stdin);
    sock_out = fileno(stdout);
#endif

/*************************
*
* ADDED FOR XTS
*
*************************/

#ifdef USE_P_SOCKET

   int shmid, debug_on = 1;
   int result, pskt_handle;
```

```
    access_ma my_sess_level;
    get_current_level(&my_sess_level);

    // Find our PSKT, using the PSKT Map DB
    result = access_pmap_db();
    if (result == PMAP_INITIALIZED)
    {
       struct passwd *pw;
       unsigned long euid;
       euid = geteuid ();
       if (pw = getpwuid (euid))
       {
          result = get_pskt_handle(pw->pw_name, my_sess_level,
&pskt_handle);
       }else{
          TMP_OUT_0("Could NOT acess pw entry\n", tmpf);
          exit(1);
       }
    }else{
       TMP_OUT_0("Could NOT access PMAP DB\n", tmpf);
       exit(1);
    }

    if (result == PMAP_FOUND)
    {
       TMP_OUT_1(tmpf_out,"Using PSKT [%d]\n", pskt_handle, tmpf);
    }else{
       TMP_OUT_0("Could NOT find PSKT\n", tmpf);
       exit(1);
    }

    // initialize access to the PSKT
    result = pskt_attach(pskt_handle);
    if (result != PSKT_INITIALIZED)
    {
       TMP_OUT_0("pskt_attach error\n", tmpf);
       exit(1);
    }

    // find the PSKT connection we are supposed to use
    result = pskt_find_connection(getpid(), &my_fd);
    if (result != PSKT_FOUND)
    {
       TMP_OUT_0("pskt_find_connection error\n", tmpf);
       exit(1);
    }else{
       TMP_OUT_1(tmpf_out,"Using PSKT fd [%d]\n", my_fd, tmpf);
    }

    // make sure we handle signals from the SSS child process
    sigset(SIGURG, handler);

    // make sure we speed up data transfer with flush calls
    // pskt_flush_required();

    sock_in = my_fd;
    sock_out = my_fd;
```

```
#endif

      l = sizeof(sa_client);
      if ((GETPEERNAME(sock_in, &sa_client, &l)) < 0) {/* mod for XTS */
/* get peername will fail if the input isn't a socket */
          perror("getpeername");
          memset(&sa_client, '\0', sizeof(sa_client));
      }

      l = sizeof(sa_server);
      if (GETSOCKNAME(sock_in, &sa_server, &l) < 0) { /* mod for XTS */
          perror("getsockname");
          fprintf(stderr, "Error getting local address\n");
          exit(1);
      }
      server_conf->port = ntohs(((struct sockaddr_in *) &sa_server)-
>sin_port);
      TMP_OUT_1(tmpf_out,"Server port [%d]\n", server_conf->port, tmpf);
      cio = ap_bcreate(ptrans, B_RDWR | B_SOCKET);
      cio->fd = sock_out;
      cio->fd_in = sock_in;
      conn = new_connection(ptrans, server_conf, cio,
                            (struct sockaddr_in *) &sa_client,
                            (struct sockaddr_in *) &sa_server, -1);


/*************************
*
* ADDED FOR XTS
*
*************************/

#ifdef USE_P_SOCKET
        result = pskt_set_aps_ready(my_fd);
        if (result != PSKT_FOUND)
        {
           TMP_OUT_0("pskt_set_aps_ready error\n", tmpf);
           exit(1);
        }
#endif            // end add for XTS

      while ((r = ap_read_request(conn)) != NULL) {

          if (r->status == HTTP_OK)
            ap_process_request(r);

          if (!conn->keepalive || conn->aborted)
            break;

          ap_destroy_pool(r->pool);
      }

      ap_bclose(cio);
    }
    exit(0);
}
```

8. The following code was added to the function *child_sub_main()* in *http_main.c* to define the SIGURG signal handler when the pseudo sockets are not being used:

```
#ifndef USE_P_SOCKET     /* added for XTS */
#if defined(SIGURG)
    signal(SIGURG, timeout);
#endif
#endif                   /* end add for XTS */
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F: MODIFICATIONS TO HEADER FILES

The following lines are the final additions to the C language configuration header file *ap_config.h.*

```
/*************************
*
* ADDED FOR XTS
*
*************************/
#elif defined(XTS)
typedef int pid_t;
#define lstat stat
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <setjmp.h>
#include <time.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/signal.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <sys/stat.h>
#define NEED_INITGROUPS       /* need groups */
#undef HAVE_MMAP              /* do not have mmap() */
#define HAVE_SHMGET           /* do support shmget() */
#undef HAVE_GMTOFF            /* no gmtoffset function */
#undef HAVE_CRYPT_H           /* no crypt.h header file */
#undef HAVE_SYS_RESOURCE_H    /* no sys/resource.h header file */
#undef USE_MMAP_SCOREBOARD    /* do not use mmap */
#undef USE_SHMGET_SCOREBOARD  /* do not use shmget */
#define USE_LONGJMP           /* use the long jump functions */
#define USE_FCNTL_SERIALIZED_ACCEPT /* use this option for fcntl() */
#define NO_MMAP               /* do not have mmap() */
#define NO_KILLPG             /* do not have killpg() */
#define NO_SETSID             /* do not have setsid() */
#define NO_USE_SIGACTION      /* do not use sigaction() */
#define NO_LINGCLOSE          /* do not allow linger of close
connections */
#define NO_GETTIMEOFDAY       /* system gettimeofday only wants one
parameter */
#define NEED_DIFFTIME         /* need difftime() */
#define S_ISLNK(mode) 0       /* no symbolic links set to 0 (false) */
#define NO_WRITEV             /* do not use writev() */

#define HTTPD_ROOT "/usr2/bersack/http"
#define SERVER_CONFIG_FILE "/usr2/bersack/http/conf/httpd.conf"
#undef PLATFORM
#define PLATFORM "XTS-300"
#define STDIN_FILENO  0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
#define SIGCHLD SIGCLD        /* redefine SIGCHLD to SIGCHD */
```

```
// set up stuff for pseudo sockets
#ifndef USE_P_SOCKET          // not using pseudo sockets
   #define PBIN getc
   #define PSIN fgets
   #define PREAD read
   #define PSIN_RET fgets
   #define PBOUT putchar
   #define PSOUT fputs
   #define PWRITE write
   #define PFLUSH fflush
   #define PFEOF feof
   #define PFERROR ferror
   #define PCLEARERR clearerr
   #define PUNGETC ungetc
   #define GETPEERNAME getpeername
   #define GETSOCKNAME getsockname
#else                         // pseudo socket definitions
   #include    "pmap_db.h"          // to find our PSKT
   #include    "pskt.h"             // to access our PSKT
   #include    "util.h"             // for check_access
   #define PBIN(fd) pbin_with_pause()
   #define PSIN(s,n,fd) psin_with_pause(s,n)
   #define PREAD(fd,s,n) psin_with_pause(s,n)
   #define PSIN_RET(s,n,fd) ((PSIN(s, n, fd)) ? s : NULL)
   #define PBOUT(c) pbout_handler (c)
   #define PSOUT(s) psout_handler (s,strlen(s))
   #define PWRITE(fd,s,n) psout_handler (s,n)
   #define PFLUSH(fd) pskt_flush(my_fd);
   #define PFEOF(fd) pskt_error(my_fd)
   #define PFERROR(fd) pskt_error(my_fd)
   #define PCLEARERR(fd) // NOOP
   #define PUNGETC(c,fd) pskt_ungetc_cli(my_fd, c)
   #define GETPEERNAME(fd,addr,len) pskt_get_peer(my_fd,addr,len)
   #define GETSOCKNAME(fd,addr,len) pskt_get_host(my_fd,addr,len)
#define TMP_OUT_0(s,fd) fwrite(s, strlen(s), 1, fd); fflush(fd);
#define TMP_OUT_1(dest,s,a1,fd) sprintf(dest, s, a1); fwrite(dest,
strlen(dest), 1, fd); fflush(fd);

   int psin_with_pause(char *s, int length);
   int pbin_with_pause();
#endif
```

APPENDIX G: TOP LEVEL MAKEFILE

This is the listing of the top-level *Makefile* generated by the *Configure* script. The platform specific lines for the XTS are indicated in bold.

```
##
##  Apache Makefile, automatically generated by Configure script.
##  Hand-edited changes will be lost if the Configure script is re-run.
##  Sources: - ./Makefile.config (via Configuration)
##           - ./Makefile.tmpl
##

MODULES= \
       modules/standard/libstandard.a

SUBDIRS=regex os/unix ap main lib modules
SUBTARGET=target_static
SHLIB_SUFFIX_NAME=
SHMOD_SUFFIX_NAME=
SHLIB_SUFFIX_LIST=
SHLIB_EXPORT_FILES=

##
##  Inherited Makefile options from Configure script
##  (Begin of automatically generated section)
##
SRCDIR=.
EXTRA_CFLAGS=
EXTRA_LDFLAGS=
EXTRA_LIBS=
EXTRA_INCLUDES=
EXTRA_DEPS=
OSDIR=$(SRCDIR)/os/unix
INCDIR=$(SRCDIR)/include
INCLUDES0=-I$(OSDIR) -I$(INCDIR)
SHELL=/bin/sh
OS=XTS
CC=cc
CPP=cc -E
TARGET=httpd
OPTIM=
CFLAGS1= -g -DXTS -I/usr2/shifflet/wip/include -DUSE_P_SOCKET
       -DUSE_HSREGEX -DUSE_EXPAT -I$(SRCDIR)/lib/expat-lite
       -DNO_DL_NEEDED
INCLUDES1=
LIBS_SHLIB=
LDFLAGS1= -lcass -lmw -lsocket -lgen
MFLAGS_STATIC=
REGLIB=regex/libregex.a
EXPATLIB=lib/expat-lite/libexpat.a
RANLIB=true
LIBS1=  -L/lib -lsocket -lmw -lgen -L/usr/lib -lcrypt
       -L/usr2/shifflet/wip/lib -lut_cass -lcass
##
##  (End of automatically generated section)
##


CFLAGS=$(OPTIM) $(CFLAGS1) $(EXTRA_CFLAGS)
LIBS=$(EXTRA_LIBS) $(LIBS1)
INCLUDES=$(INCLUDES1) $(INCLUDES0) $(EXTRA_INCLUDES)
LDFLAGS=$(LDFLAGS1) $(EXTRA_LDFLAGS)
```

```
OBJS= \
  modules.o \
  $(MODULES) \
  main/libmain.a \
  $(OSDIR)/libos.a \
  ap/libap.a

.c.o:
	$(CC) -c $(INCLUDES) $(CFLAGS) $<

# Used to generate import library for OS/2
.SUFFIXES: .def
.def.a:
	emximp -o $@ $<

all: Configuration $(TARGET)

Configuration: Configuration.tmpl
	@echo "++ File 'Configuration' older than 'Configuration.tmpl'," 
	@echo "++ or still doesn't exist. Please consider copying
'Configuration.tmpl'"
	@echo "++ to 'Configuration', editing and rerunning 'Configure'."
	@echo "++ If not, you will at least have to touch 'Configuration'."
	@false

$(TARGET): $(EXTRA_DEPS) $(SUBTARGET)

target_static: subdirs modules.o
	$(CC) -c $(INCLUDES) $(CFLAGS) buildmark.c
	$(CC) $(CFLAGS) $(LDFLAGS) $(LDFLAGS_SHLIB_EXPORT) \
	      -o $(TARGET) buildmark.o $(OBJS) $(REGLIB) $(EXPATLIB) $(LIBS)

target_compile_only: subdirs modules.o
	$(CC) -c $(INCLUDES) $(CFLAGS) buildmark.c

target_shared: $(SHCORE_IMPLIB) $(SHARED_CORE_EP)
lib$(TARGET).$(SHLIB_SUFFIX_NAME)
	$(CC) $(INCLUDES) $(CFLAGS) $(LDFLAGS) $(LDFLAGS_SHLIB_EXPORT) \
	      -o $(TARGET) -DSHARED_CORE_BOOTSTRAP main/http_main.c \
	      ap/libap.a $(LIBS) $(SHCORE_IMPLIB)

lib$(TARGET).ep: lib$(TARGET).$(SHLIB_SUFFIX_NAME)
	$(CC) $(INCLUDES) $(CFLAGS) $(LDFLAGS) $(LDFLAGS_SHLIB_EXPORT) \
	      -o lib$(TARGET).ep -DSHARED_CORE_TIESTATIC main/http_main.c\
	       -L. -l$(TARGET) $(LIBS)

lib$(TARGET).$(SHLIB_SUFFIX_NAME): subdirs modules.o
	$(CC) -c $(INCLUDES) $(CFLAGS) buildmark.c
	$(LD_SHLIB) $(LDFLAGS_SHLIB) -o lib$(TARGET).$(SHLIB_SUFFIX_NAME)
buildmark.o $(OBJS) $(REGLIB) $(EXPATLIB) $(LD_SHCORE_DEF) $(LD_SHCORE_LIBS)
	@if [ ".$(SHLIB_SUFFIX_LIST)" != . ]; then \
	      rm -f lib$(TARGET).$(SHLIB_SUFFIX_NAME).*; \
	      for suffix in $(SHLIB_SUFFIX_LIST) ""; do \
	            [ ".$$suffix" = . ] && continue; \
	          echo "ln lib$(TARGET).$(SHLIB_SUFFIX_NAME)
lib$(TARGET).$(SHLIB_SUFFIX_NAME).$$suffix"; \
	          ln lib$(TARGET).$(SHLIB_SUFFIX_NAME)
lib$(TARGET).$(SHLIB_SUFFIX_NAME).$$suffix; \
	            done; \
	fi

subdirs:
	@for i in $(SUBDIRS); do \
	      echo "===> $(SDP)$$i"; \
	      case ".$(OS)" in \
```

```
                .OS390 | .TPF) ( cd $$i && $(MAKE) SDP='$(SDP)' ) || exit 1;; \
                       *) ( cd $$i && $(MAKE) $(MFLAGS_STATIC) SDP='$(SDP)'
CC='$(CC)' AUX_CFLAGS='$(CFLAGS)' RANLIB='$(RANLIB)') || exit 1;; \
              esac; \
              echo "<=== $(SDP)$$i"; \
       done

support: support-dir

support-dir:
       @echo "===> $(SDP)support"; \
       cd support; $(MAKE) $(MFLAGS_STATIC) SDP='$(SDP)' CC='$(CC)'
AUX_CFLAGS='$(CFLAGS)' RANLIB='$(RANLIB)' || exit 1; \
       echo "<=== $(SDP)support"

clean:
       -rm -f $(TARGET) lib$(TARGET).* *.o
       @for i in $(SUBDIRS); do \
              echo "===> $(SDP)$$i"; \
              ( cd $$i && $(MAKE) $(MFLAGS_STATIC) SDP='$(SDP)' $@ ) || exit 1;
\
              echo "<=== $(SDP)$$i"; \
       done

distclean:
       -rm -f $(TARGET) lib$(TARGET).* *.o
       @for i in $(SUBDIRS); do \
              echo "===> $(SDP)$$i"; \
              ( cd $$i && $(MAKE) $(MFLAGS_STATIC) SDP='$(SDP)' $@ ) || exit 1;
\
              echo "<=== $(SDP)$$i"; \
       done
       -rm -f include/ap_config_auto.h
       -rm -f modules.c
       -rm -f modules/Makefile
       -rm -f regex/Makefile
       -rm -f lib/Makefile
       -rm -f Makefile.config
       -rm -f Makefile

install:
       @echo "++ Sorry, no installation procedure available at this level."
       @echo "++ Go to the parent directory for an 'install' target."

# We really don't expect end users to use this rule.  It works only with
# gcc, and rebuilds Makefile.tmpl.  You have to re-run Configure after
# using it.
depend:
       cp Makefile.tmpl Makefile.tmpl.bak \
           && sed -ne '1,/^# DO NOT REMOVE/p' Makefile.tmpl > Makefile.new \
           && gcc -MM $(INCLUDES) $(CFLAGS) *.c >> Makefile.new \
           && sed -e '1,$$s: $(INCDIR)/: $$(INCDIR)/:g' \
                  -e '1,$$s: $(OSDIR)/: $$(OSDIR)/:g' Makefile.new \
              > Makefile.tmpl \
           && rm Makefile.new
       for i in $(SUBDIRS); do \
           ( cd $$i && $(MAKE) CC='$(CC)' AUX_CFLAGS='$(CFLAGS)'
RANLIB='$(RANLIB)' depend ) || exit 1; \
       done

#Dependencies

$(OBJS): Makefile subdirs

# DO NOT REMOVE
buildmark.o: buildmark.c include/ap_config.h include/ap_mmn.h \
```

```
 include/ap_config_auto.h os/unix/os.h include/ap_ctype.h \
 include/hsregex.h include/httpd.h include/alloc.h include/buff.h \
 include/ap.h include/util_uri.h
modules.o: modules.c include/httpd.h include/ap_config.h \
 include/ap_mmn.h include/ap_config_auto.h os/unix/os.h \
 include/ap_ctype.h include/hsregex.h include/alloc.h include/buff.h \
 include/ap.h include/util_uri.h include/http_config.h
```

APPENDIX H: SERVER CONFIGURATION FILE

This is the final server configuration file, *httpd.conf*, used with this implementation. The changes are in bold.

```
#
# Based upon the NCSA server configuration files originally by Rob McCool.
#
# This is the main Apache server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://www.apache.org/docs/> for detailed information about
# the directives.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# After this file is processed, the server will look for and process
# @@ServerRoot@@/conf/srm.conf and then @@ServerRoot@@/conf/access.conf
# unless you have overridden these with ResourceConfig and/or
# AccessConfig directives here.
#
# The configuration directives are grouped into three basic sections:
#  1. Directives that control the operation of the Apache server process as a
#     whole (the 'global environment').
#  2. Directives that define the parameters of the 'main' or 'default' server,
#     which responds to requests that aren't handled by a virtual host.
#     These directives also provide default values for the settings
#     of all virtual hosts.
#  3. Settings for virtual hosts, which allow Web requests to be sent to
#     different IP addresses or hostnames and have them handled by the
#     same Apache server process.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "logs/foo.log"
# with ServerRoot set to "/usr/local/apache" will be interpreted by the
# server as "/usr/local/apache/logs/foo.log".
#

### Section 1: Global Environment
#
# The directives in this section affect the overall operation of Apache,
# such as the number of concurrent requests it can handle or where it
# can find its configuration files.
#

#
# ServerType is either inetd, or standalone.  Inetd mode is only supported on
# Unix platforms.
#
ServerType inetd

#
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# NOTE!  If you intend to place this on an NFS (or otherwise network)
# mounted filesystem then please read the LockFile documentation
# (available at <URL:http://www.apache.org/docs/mod/core.html#lockfile>);
# you will save yourself a lot of trouble.
```

```
#
# Do NOT add a slash at the end of the directory path.
#
ServerRoot /usr2/bersack/http

#
# The LockFile directive sets the path to the lockfile used when Apache
# is compiled with either USE_FCNTL_SERIALIZED_ACCEPT or
# USE_FLOCK_SERIALIZED_ACCEPT. This directive should normally be left at
# its default value. The main reason for changing it is if the logs
# directory is NFS mounted, since the lockfile MUST BE STORED ON A LOCAL
# DISK. The PID of the main server process is automatically appended to
# the filename.
#
#LockFile logs/accept.lock

#
# PidFile: The file in which the server should record its process
# identification number when it starts.
#
PidFile /tmp/httpd.pid

#
# ScoreBoardFile: File used to store internal server process information.
# Not all architectures require this.  But if yours does (you'll know because
# this file will be  created when you run Apache) then you *must* ensure that
# no two invocations of Apache share the same scoreboard file.
#
ScoreBoardFile /tmp/apache_runtime_status

#
# In the standard configuration, the server will process this file,
# srm.conf, and access.conf in that order.  The latter two files are
# now distributed empty, as it is recommended that all directives
# be kept in a single file for simplicity.  The commented-out values
# below are the built-in defaults.  You can have the server ignore
# these files altogether by using "/dev/null" (for Unix) or
# "nul" (for Win32) for the arguments to the directives.
#
#ResourceConfig conf/srm.conf
#AccessConfig conf/access.conf

#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 10

#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive Off

#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100

#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 300
```

```
#
# Server-pool size regulation.  Rather than making you guess how many
# server processes you need, Apache dynamically adapts to the load it
# sees --- that is, it tries to maintain enough server processes to
# handle the current load, plus a few spare servers to handle transient
# load spikes (e.g., multiple simultaneous requests from a single
# Netscape browser).
#
# It does this by periodically checking how many servers are waiting
# for a request.  If there are fewer than MinSpareServers, it creates
# a new spare.  If there are more than MaxSpareServers, some of the
# spares die off.  The default values are probably OK for most sites.
#
MinSpareServers 5
MaxSpareServers 10

#
# Number of servers to start initially --- should be a reasonable ballpark
# figure.
#
StartServers 0

#
# Limit on total number of servers running, i.e., limit on the number
# of clients who can simultaneously connect --- if this limit is ever
# reached, clients will be LOCKED OUT, so it should NOT BE SET TOO LOW.
# It is intended mainly as a brake to keep a runaway server from taking
# the system with it as it spirals down...
#
MaxClients 150

#
# MaxRequestsPerChild: the number of requests each child process is
# allowed to process before the child dies.  The child will exit so
# as to avoid problems after prolonged use when Apache (and maybe the
# libraries it uses) leak memory or other resources.  On most systems, this
# isn't really needed, but a few (such as Solaris) do have notable leaks
# in the libraries. For these platforms, set to something like 10000
# or so; a setting of 0 means unlimited.
#
# NOTE: This value does not include keepalive requests after the initial
#       request per connection. For example, if a child process handles
#       an initial request and 10 subsequent "keptalive" requests, it
#       would only count as 1 request towards this limit.
#
MaxRequestsPerChild 0

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, in addition to the default. See also the <VirtualHost>
# directive.
#
#Listen 3000
#Listen 12.34.56.78:80

#
# BindAddress: You can support virtual hosts with this option. This directive
# is used to tell the server which IP address to listen to. It can either
# contain "*", an IP address, or a fully qualified Internet domain name.
# See also the <VirtualHost> and Listen directives.
#
#BindAddress *

#
```

```
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding 'LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Please read the file README.DSO in the Apache 1.3 distribution for more
# details about the DSO mechanism and run 'httpd -l' for the list of already
# built-in (statically linked and thus always available) modules in your httpd
# binary.
#
# Note: The order is which modules are loaded is important.  Don't change
# the order below without expert advice.
#
# Example:
# LoadModule foo_module libexec/mod_foo.so

#
# ExtendedStatus controls whether Apache will generate "full" status
# information (ExtendedStatus On) or just basic information (ExtendedStatus
# Off) when the "server-status" handler is called. The default is Off.
#
#ExtendedStatus On

# See if we can include the module mod_include
# for server side includes
# AddModule mod_include

### Section 2: 'Main' server configuration
#
# The directives in this section set up the values used by the 'main'
# server, which responds to any requests that aren't handled by a
# <VirtualHost> definition.  These values also provide defaults for
# any <VirtualHost> containers you may define later in the file.
#
# All of these directives may appear inside <VirtualHost> containers,
# in which case these default settings will be overridden for the
# virtual host being defined.
#

#
# If your ServerType directive (set earlier in the 'Global Environment'
# section) is set to "inetd", the next few directives don't have any
# effect since their settings are defined by the inetd configuration.
# Skip ahead to the ServerAdmin directive.
#

#
# Port: The port to which the standalone server listens. For
# ports < 1023, you will need httpd to be run as root initially.
#
Port 2000

#
# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
# User/Group: The name (or #number) of the user/group to run httpd as.
#  . On SCO (ODT 3) use "User nouser" and "Group nogroup".
#  . On HPUX you may not be able to use shared memory as nobody, and the
#    suggested workaround is to create a user www and use that user.
#  NOTE that some kernels refuse to setgid(Group) or semctl(IPC_SET)
#  when the value of (unsigned)Group is above 60000;
#  don't use Group #-1 on these systems!
#
#User nobody
```

```
User bersack
#Group #-1
Group other

#
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents.
#
ServerAdmin bersack@holmes

#
# ServerName allows you to set a host name which is sent back to clients for
# your server if it's different than the one the program would get (i.e., use
# "www" instead of the host's real name).
#
# Note: You cannot just invent host names and hope they work. The name you
# define here must be a valid DNS name for your host. If you don't understand
# this, ask your network administrator.
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address (e.g., http://123.45.67.89/)
# anyway, and this will make redirections work in a sensible way.
#
ServerName 131.120.10.99

#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "/usr2/bersack/http/htdocs"

#
# Each directory to which Apache has access, can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# permissions.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#

#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "/usr2/bersack/http/htdocs">

#
# This may also be "None", "All", or any combination of "Indexes",
# "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
    Options Indexes FollowSymLinks MultiViews
```

```
#
# This controls which options the .htaccess files in directories can
# override. Can also be "All", or any combination of "Options", "FileInfo",
# "AuthConfig", and "Limit"
#
    AllowOverride None

#
# Controls who can get stuff from this server.
#
    Order allow,deny
    Allow from all
</Directory>

#
# Now changes so server side includes can be used
#
AddType text/html .shtml
AddHandler server-parsed .shtml

<Directory "/usr2/bersack/http/htdocs/mls_test">
    Options +Includes
</Directory>

#
# UserDir: The name of the directory which is appended onto a user's home
# directory if a ~user request is received.
#
<IfModule mod_userdir.c>
    UserDir public_html
</IfModule>


#
# Control access to UserDir directories.  The following is an example
# for a site where these directories are restricted to read-only.
#
#<Directory /home/*/public_html>
#    AllowOverride FileInfo AuthConfig Limit
#    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
#    <Limit GET POST OPTIONS PROPFIND>
#        Order allow,deny
#        Allow from all
#    </Limit>
#    <LimitExcept GET POST OPTIONS PROPFIND>
#        Order deny,allow
#        Deny from all
#    </LimitExcept>
#</Directory>

#
# DirectoryIndex: Name of the file or files to use as a pre-written HTML
# directory index.  Separate multiple entries with spaces.
#
<IfModule mod_dir.c>
    DirectoryIndex index.html
</IfModule>

#
# AccessFileName: The name of the file to look for in each directory
# for access control information.
#
AccessFileName .htaccess
```

```
#
# The following lines prevent .htaccess files from being viewed by
# Web clients.  Since .htaccess files often contain authorization
# information, access is disallowed for security reasons.  Comment
# these lines out if you want Web visitors to see the contents of
# .htaccess files.  If you change the AccessFileName directive above,
# be sure to make the corresponding changes here.
#
# Also, folks tend to use names such as .htpasswd for password
# files, so this will protect those as well.
#
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>

#
# CacheNegotiatedDocs: By default, Apache sends "Pragma: no-cache" with each
# document that was negotiated on the basis of content. This asks proxy
# servers not to cache the document. Uncommenting the following line disables
# this behavior, and proxies will be allowed to cache the documents.
#
#CacheNegotiatedDocs

#
# UseCanonicalName:  (new for 1.3)  With this setting turned on, whenever
# Apache needs to construct a self-referencing URL (a URL that refers back
# to the server the response is coming from) it will use ServerName and
# Port to form a "canonical" name.  With this setting off, Apache will
# use the hostname:port that the client supplied, when possible.  This
# also affects SERVER_NAME and SERVER_PORT in CGI scripts.
#
UseCanonicalName On

#
# TypesConfig describes where the mime.types file (or equivalent) is
# to be found.
#
<IfModule mod_mime.c>
    TypesConfig conf/mime.types
</IfModule>

#
# DefaultType is the default MIME type the server will use for a document
# if it cannot otherwise determine one, such as from filename extensions.
# If your server contains mostly text or HTML documents, "text/plain" is
# a good value.  If most of your content is binary, such as applications
# or images, you may want to use "application/octet-stream" instead to
# keep browsers from trying to display binary files as though they are
# text.
#
DefaultType text/plain

#
# The mod_mime_magic module allows the server to use various hints from the
# contents of the file itself to determine its type.  The MIMEMagicFile
# directive tells the module where the hint definitions are located.
# mod_mime_magic is not part of the default server (you have to add
# it yourself with a LoadModule [see the DSO paragraph in the 'Global
# Environment' section], or recompile the server and include mod_mime_magic
# as part of the configuration), so it's enclosed in an <IfModule> container.
# This means that the MIMEMagicFile directive will only be processed if the
# module is part of the server.
#
<IfModule mod_mime_magic.c>
    MIMEMagicFile conf/magic
```

```
</IfModule>

#
# HostnameLookups: Log the names of clients or just their IP addresses
# e.g., www.apache.org (on) or 204.62.129.132 (off).
# The default is off because it'd be overall better for the net if people
# had to knowingly turn this feature on, since enabling it means that
# each client request will result in AT LEAST one lookup request to the
# nameserver.
#
HostnameLookups Off

#
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
ErrorLog /tmp/error_log

#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel debug

#
# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
#
#LogFormat  "%h  %l  %u  %t  \"%r\"  %>s  %b  \"%{Referer}i\"  \"%{User-Agent}i\""
combined
#LogFormat "%h %l %u %t \"%r\" %>s %b" common
#LogFormat "%{Referer}i -> %U" referer
#LogFormat "%{User-agent}i" agent

#
# The location and format of the access logfile (Common Logfile Format).
# If you do not define any access logfiles within a <VirtualHost>
# container, they will be logged here.  Contrariwise, if you *do*
# define per-<VirtualHost> access logfiles, transactions will be
# logged therein and *not* in this file.
#
#CustomLog logs/access_log common

#
# If you would like to have agent and referer logfiles, uncomment the
# following directives.
#
#CustomLog logs/referer_log referer
#CustomLog logs/agent_log agent

#
# If you prefer a single logfile with access, agent, and referer information
# (Combined Logfile Format) you can use the following directive.
#
#CustomLog logs/access_log combined

#
# Optionally add a line containing the server version and virtual host
# name to server-generated pages (error documents, FTP directory listings,
# mod_status and mod_info output etc., but not CGI generated documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
```

```
# Set to one of:  On | Off | EMail
#
ServerSignature On

#
# Aliases: Add here as many aliases as you need (with no limit). The format is
# Alias fakename realname
#
<IfModule mod_alias.c>

    #
    # Note that if you include a trailing / on fakename then the server will
    # require it to be present in the URL.  So "/icons" isn't aliased in this
    # example, only "/icons/"..
    #
    Alias /icons/ "@@ServerRoot@@/icons/"

    <Directory "@@ServerRoot@@/icons">
        Options Indexes MultiViews
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

    #
    # ScriptAlias: This controls which directories contain server scripts.
    # ScriptAliases are essentially the same as Aliases, except that
    # documents in the realname directory are treated as applications and
    # run by the server when requested rather than as documents sent to the
client.
    # The same rules about trailing "/" apply to ScriptAlias directives as to
    # Alias.
    #
    ScriptAlias /cgi-bin/ "@@ServerRoot@@/cgi-bin/"

    #
    # "@@ServerRoot@@/cgi-bin" should be changed to whatever your ScriptAliased
    # CGI directory exists, if you have that configured.
    #
    <Directory "@@ServerRoot@@/cgi-bin">
        AllowOverride None
        Options None
        Order allow,deny
        Allow from all
    </Directory>

</IfModule>
# End of aliases.

#
# Redirect allows you to tell clients about documents which used to exist in
# your server's namespace, but do not anymore. This allows you to tell the
# clients where to look for the relocated document.
# Format: Redirect old-URI new-URL
#

#
# Directives controlling the display of server-generated directory listings.
#
<IfModule mod_autoindex.c>

    #
    # FancyIndexing is whether you want fancy directory indexing or standard
    #
    IndexOptions FancyIndexing
```

```
#
# AddIcon* directives tell the server which icon to show for different
# files or filename extensions.  These are only displayed for
# FancyIndexed directories.
#
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core

AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

#
# DefaultIcon is which icon to show for files which do not have an icon
# explicitly set.
#
DefaultIcon /icons/unknown.gif

#
# AddDescription allows you to place a short description after a file in
# server-generated indexes.  These are only displayed for FancyIndexed
# directories.
# Format: AddDescription "description" filename
#
#AddDescription "GZIP compressed document" .gz
#AddDescription "tar archive" .tar
#AddDescription "GZIP compressed tar archive" .tgz

#
# ReadmeName is the name of the README file the server will look for by
# default, and append to directory listings.
#
# HeaderName is the name of a file which should be prepended to
# directory indexes.
#
# If MultiViews are amongst the Options in effect, the server will
# first look for name.html and include it if found.  If name.html
# doesn't exist, the server will then look for name.txt and include
# it as plaintext if found.
#
ReadmeName README
HeaderName HEADER

#
```

```
    # IndexIgnore is a set of filenames which directory indexing should ignore
    # and not include in the listing.  Shell-style wildcarding is permitted.
    #
    IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t

</IfModule>
# End of indexing directives.

#
# Document types.
#
<IfModule mod_mime.c>

    #
    # AddEncoding allows you to have certain browsers (Mosaic/X 2.1+) uncompress
    # information on the fly. Note: Not all browsers support this.
    # Despite the name similarity, the following Add* directives have nothing
    # to do with the FancyIndexing customization directives above.
    #
    AddEncoding x-compress Z
    AddEncoding x-gzip gz tgz

    #
    # AddLanguage allows you to specify the language of a document. You can
    # then use content negotiation to give a browser a file in a language
    # it can understand.
    #
    # Note 1: The suffix does not have to be the same as the language
    # keyword --- those with documents in Polish (whose net-standard
    # language code is pl) may wish to use "AddLanguage pl .po" to
    # avoid the ambiguity with the common suffix for perl scripts.
    #
    # Note 2: The example entries below illustrate that in quite
    # some cases the two character 'Language' abbriviation is not
    # identical to the two character 'Country' code for its country,
    # E.g. 'Danmark/dk' versus 'Danish/da'.
    #
    # Note 3: In the case of 'ltz' we violate the RFC by using a three char
    # specifier. But there is 'work in progress' to fix this and get
    # the reference data for rfc1766 cleaned up.
    #
    # Danish (da) - Dutch (nl) - English (en) - Estonian (ee)
    # French (fr) - German (de) - Greek-Modern (el)
    # Italian (it) - Portugese (pt) - Luxembourgeois* (ltz)
    # Spanish (es) - Swedish (sv) - Catalan (ca) - Czech(cz)
    # Polish (pl) - Brazilian Portuguese (pt-br) - Japanese (ja)
    #
    AddLanguage da .dk
    AddLanguage nl .nl
    AddLanguage en .en
    AddLanguage et .ee
    AddLanguage fr .fr
    AddLanguage de .de
    AddLanguage el .el
    AddLanguage it .it
    AddLanguage ja .ja
    AddCharset ISO-2022-JP .jis
    AddLanguage pl .po
    AddCharset ISO-8859-2 .iso-pl
    AddLanguage pt .pt
    AddLanguage pt-br .pt-br
    AddLanguage ltz .lu
    AddLanguage ca .ca
    AddLanguage es .es
    AddLanguage sv .se
    AddLanguage cz .cz
```

```
    # LanguagePriority allows you to give precedence to some languages
    # in case of a tie during content negotiation.
    #
    # Just list the languages in decreasing order of preference. We have
    # more or less alphabetized them here. You probably want to change this.
    #
    <IfModule mod_negotiation.c>
        LanguagePriority en da nl et fr de el it ja pl pt pt-br ltz ca es sv
    </IfModule>

    #
    # AddType allows you to tweak mime.types without actually editing it, or to
    # make certain files to be certain types.
    #
    # For example, the PHP 3.x module (not part of the Apache distribution - see
    # http://www.php.net) will typically use:
    #
    #AddType application/x-httpd-php3 .php3
    #AddType application/x-httpd-php3-source .phps
    #
    # And for PHP 4.x, use:
    #
    #AddType application/x-httpd-php .php
    #AddType application/x-httpd-php-source .phps

    AddType application/x-tar .tgz

    #
    # AddHandler allows you to map certain file extensions to "handlers",
    # actions unrelated to filetype. These can be either built into the server
    # or added with the Action command (see below)
    #
    # If you want to use server side includes, or CGI outside
    # ScriptAliased directories, uncomment the following lines.
    #
    # To use CGI scripts:
    #
    #AddHandler cgi-script .cgi

    #
    # To use server-parsed HTML files
    #
    #AddType text/html .shtml
    #AddHandler server-parsed .shtml

    #
    # Uncomment the following line to enable Apache's send-asis HTTP file
    # feature
    #
    #AddHandler send-as-is asis

    #
    # If you wish to use server-parsed imagemap files, use
    #
    #AddHandler imap-file map

    #
    # To enable type maps, you might want to use
    #
    #AddHandler type-map var

</IfModule>
# End of document types.

#
```

```
# Action lets you define media types that will execute a script whenever
# a matching file is called. This eliminates the need for repeated URL
# pathnames for oft-used CGI file processors.
# Format: Action media/type /cgi-script/location
# Format: Action handler-name /cgi-script/location
#

#
# MetaDir: specifies the name of the directory in which Apache can find
# meta information files. These files contain additional HTTP headers
# to include when sending the document
#
#MetaDir .web

#
# MetaSuffix: specifies the file name suffix for the file containing the
# meta information.
#
#MetaSuffix .meta

#
# Customizable error response (Apache style)
#  these come in three flavors
#
#    1) plain text
#ErrorDocument 500 "The server made a boo boo.
#  n.b.  the (") marks it as text, it does not get output
#
#    2) local redirects
#ErrorDocument 404 /missing.html
#  to redirect to local URL /missing.html
#ErrorDocument 404 /cgi-bin/missing_handler.pl
#  N.B.: You can redirect to a script or a document using server-side-includes.
#
#    3) external redirects
#ErrorDocument 402 http://some.other_server.com/subscription_info.html
#  N.B.: Many of the environment variables associated with the original
#  request will *not* be available to such a script.

#
# Customize behaviour based on the browser
#
<IfModule mod_setenvif.c>

    #
    # The following directives modify normal HTTP response behavior.
    # The first directive disables keepalive for Netscape 2.x and browsers that
    # spoof it. There are known problems with these browser implementations.
    # The second directive is for Microsoft Internet Explorer 4.0b2
    # which has a broken HTTP/1.1 implementation and does not properly
    # support keepalive when it is used on 301 or 302 (redirect) responses.
    #
    BrowserMatch "Mozilla/2" nokeepalive
    BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

    #
    # The following directive disables HTTP/1.1 responses to browsers which
    # are in violation of the HTTP/1.0 spec by not being able to grok a
    # basic 1.1 response.
    #
    BrowserMatch "RealPlayer 4\.0" force-response-1.0
    BrowserMatch "Java/1\.0" force-response-1.0
    BrowserMatch "JDK/1\.0" force-response-1.0

</IfModule>
```

```
#
# Allow server status reports, with the URL of http://servername/server-status
# Change the ".your_domain.com" to match your domain to enable.
#
#<Location /server-status>
#    SetHandler server-status
#    Order deny,allow
#    Deny from all
#    Allow from .your_domain.com
#</Location>

#
# Allow remote server configuration reports, with the URL of
#  http://servername/server-info (requires that mod_info.c be loaded).
# Change the ".your_domain.com" to match your domain to enable.
#
#<Location /server-info>
#    SetHandler server-info
#    Order deny,allow
#    Deny from all
#    Allow from .your_domain.com
#</Location>

#
# There have been reports of people trying to abuse an old bug from pre-1.1
# days.  This bug involved a CGI script distributed as a part of Apache.
# By uncommenting these lines you can redirect these attacks to a logging
# script on phf.apache.org.  Or, you can record them yourself, using the script
# support/phf_abuse_log.cgi.
#
#<Location /cgi-bin/phf*>
#    Deny from all
#    ErrorDocument 403 http://phf.apache.org/phf_abuse_log.cgi
#</Location>

#
# Proxy Server directives. Uncomment the following lines to
# enable the proxy server:
#
#<IfModule mod_proxy.c>
    #ProxyRequests On
    #
    #<Directory proxy:*>
    #    Order deny,allow
    #    Deny from all
    #    Allow from .your_domain.com
    #</Directory>

    #
    # Enable/disable the handling of HTTP/1.1 "Via:" headers.
    #  ("Full" adds the server version; "Block" removes all outgoing Via:
headers)
    # Set to one of: Off | On | Full | Block
    #
    #ProxyVia On

    #
    # To enable the cache as well, edit and uncomment the following lines:
    # (no cacheing without CacheRoot)
    #
    #CacheRoot "@@ServerRoot@@/proxy"
    #CacheSize 5
    #CacheGcInterval 4
    #CacheMaxExpire 24
    #CacheLastModifiedFactor 0.1
    #CacheDefaultExpire 1
```

```
    #NoCache a_domain.com another_domain.edu joes.garage_sale.com

#</IfModule>
# End of proxy directives.

### Section 3: Virtual Hosts
#
# VirtualHost: If you want to maintain multiple domains/hostnames on your
# machine you can setup VirtualHost containers for them.
# Please see the documentation at <URL:http://www.apache.org/docs/vhosts/>
# for further details before you try to setup virtual hosts.
# You may use the command line option '-S' to verify your virtual host
# configuration.

#
# If you want to use name-based virtual hosts you need to define at
# least one IP address (and port number) for them.
#
#NameVirtualHost 12.34.56.78:80
#NameVirtualHost 12.34.56.78

#
# VirtualHost example:
# Almost any Apache directive may go into a VirtualHost container.
#
#<VirtualHost ip.address.of.host.some_domain.com>
#    ServerAdmin webmaster@host.some_domain.com
#    DocumentRoot /www/docs/host.some_domain.com
#    ServerName host.some_domain.com
#    ErrorLog logs/host.some_domain.com-error_log
#    CustomLog logs/host.some_domain.com-access_log common
#</VirtualHost>

#<VirtualHost _default_:*>
#</VirtualHost>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I: APACHE MODULES

Apache Add-On Modules			
Module Name	**Type**	**Description**	**Security Related**
mod_access	Access Control	Provides access control based on client hostname or IP address	Y
mod_actions	Dynamic Content	Provides for executing CGI scripts based on media type or request method. Module lets you run CGI scripts whenever a file of a certain type is requested. This makes it much easier to execute scripts that process files	Y
mod_alias	URL mapping	Provides for mapping different parts of the host filesystem in the document tree, and for URL redirection	N
mod_asis	HTTP response	Provides for .asis files. Any document with mime type httpd/send-as-is will be processed by this module. Allow file types to be defined such that Apache sends them without adding HTTP headers	N
mod_auth	Access Control	Provides for user authentication using textual files	Y
mod_auth_anon	Access Control	Allows "anonymous" user access to authenticated areas	Y
mod_auth_db	Access Control	For user authentication using Berkeley DB files	Y
mod_auth_dbm	Access Control	Provides for user authentication using DBM files	Y
mod_auth_digest	Access Control	Provides for user authentication using MD5 Digest Authentication	Y
mod_autoindex	Directory Handling	Provides for automatic directory indexing	Y
mod_browser	**Obsolete**	**Obsolete in 1.3.***	
mod_cern_meta	HTTP response	Provides for CERN httpd metafile semantics; meta information is descriptive information contained in the file	Y

mod_cgi	Dynamic Content	Provides for execution of Common Gateway Interface (CGI) scripts	Y
mod_cookies	**Obsolete**	**Obsolete in 1.3.***	
mod_digest	Access Control	Provides for user authentication using MD5 Digest Authentication	Y
mod_dir	Directory Handling	Provides for "trailing slash" redirects and serving directory index files	N
mod_dld	**Obsolete**	**Obsolete in 1.3.***	
mod_dll	**Obsolete**	**Obsolete in 1.3.***	
mod_env	Environment	Provides for passing environment variables to CGI/SSI scripts	Y
mod_example	Development	Provides example code for writing user defined modules	N
mod_expires	HTTP response	Provides for generation of Expires HTTP headers (refers to document validity and persistence	N
mod_headers	HTTP response	Provides for customization of user headers	Y
mod_imap	Misc.	Processes *.map files, imagemap files	N
mod_include	Dynamic Content	Provides a handler for parsing files before they are sent to the client	Y
mod_info	Internal Content	Provides for comprehensive server configuration	Y
mod_isapi	Dynamic Content	Implements Internet Server extension API for Windows	
mod_log_agent	**Obsolete**	**Obsolete in 1.3.***	
mod_log_common	**Obsolete**	**Obsolete in 1.3.***	
mod_log_config	Logging	Provides for logging of requests made to the server	Y
mod_log_referer	**Obsolete**	**Obsolete in 1.3.***	
mod_mime	Content type	Provides for determining types of files from the filename (determine meta information about documents)	N
mod_mime_magic	Content type	Provides for determining the MIME type of a file, like the file	N

		command in UNIX	
mod_mmap_static	Misc.	Provides mmapping() of a statically configured list of frequently requested but not changed files (experimental)	N
mod_negotiation	Content type	Provides for content negotiation, selection of the document that best matches the clients capabilities	N
mod_proxy	Misc.	Implements a proxy/cache for Apache	Y
mod_rewrite	URL mapping	Provides a rule-based rewriting engine to rewrite requested URLs on the fly	N
mod_setenvif	Environment	Provides the ability to set environment variables based upon attributes of the request	Y
mod_so	Misc.	Provides for loading of executable code and modules into the server at start-up or restart time (experimental)	N
mod_speling	URL mapping	Attempts to correct misspellings of URLs	Y
mod_status	Internal Content	Provides information on server status, activity, and performance	Y
mod_userdir	URL mapping	Provides for user specific directories, allows users to home web pages	Y
mod_usertrack	Logging	Uses cookies to provide for a clickstream log of user activity on a site	Y
mod_vhost_alias	URL mapping	Creates dynamically configured virtual hosts by allowing the IP address and/or Host: header of the HTTP request to be used as part of the pathname to determine what files to serve	Y

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Apache Software Foundation, "Apache Project", http://httpd.apache.org

2. The Apache Software Foundation, "About the Apache HTTP Server Project", http://www.apache.org/ABOUT_APACHE.html, February 1999.

3. Wang Government Services, Inc., 7900 Westpark Drive, McLean, VA 22102, *XTS-300 User's Manual*, STOP 4.4.2 Version, March 1998.

4. World Wide Web Consortium, "HTTP – HyperText Transfer Protocol", http://www.w3c.org/Protocols/Overview.html, v 1.186 2000/07/06.

5. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., "HyperText Transfer Protocol – HTTP/1.1", Internet Engineering Task Force Request For Comment 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

6. Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", Internet Engineering Task Force Request For Comment 2396, January 1997, http://www.ietf.org/rfc/rfc2396.txt

7. Berners-Lee, T., Fielding, R., and Frystyk, H., "HyperText Transfer Protocol – HTTP/1.0", Internet Engineering Task Force Request For Comment 1945, May 1996, http://www.ietf.org/rfc/rfc1945.txt

8. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., and Berners-Lee, T., "HyperText Transfer Protocol – HTTP/1.1", Internet Engineering Task Force Request For Comment 2068, January 1997, http://www.ietf.org/rfc/rfc2068.txt

9. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., and Berners-Lee, T., "Use and Interpretation of HTTP Version Numbers", Internet Engineering Task Force Request For Comment 2145, May 1997, http://www.ietf.org/rfc/rfc2145.txt

10. Holtman, K., Mutz, A. "Transparent Content Negotiation in HTTP", Internet Engineering Task Force Request For Comment 2295, March 1998, http://www.ietf.org/rfc/rfc2295.txt

11. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication", Internet Engineering Task Force Request For Comment 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt

12. World Wide Web Consortium, "HTTP Activity Statement", http://www.w3.org/Protocols/Activity.html, October 2000.

13. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., "HyperText Transfer Protocol – HTTP/1.1", Internet Engineering Task Force Request For Comment 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt, Section 14.23.

14. Xie, Geoffrey, "CS4550 Computer Networks II - Summer 2000 Lecture Notes", Naval Postgraduate School, Monterey, CA, Summer 2000.

15. Wainwright, Peter, *Professional Apache*, Wrox Press Ltd., 1999.

16. Wang Government Services, Inc., 7900 Westpark Drive, McLean, VA 22102, "XTS-300 Release 4.4.2", http://www.radium.ncsc.mil/tpep/epl/entries/CSC-EPL-92-003-D.html, March 31, 1998.

17. Bell, D. E., and LaPadula, L., *Secure Computer Systems: mathematical Foundations and Model,* M74-244, MITRE Corp., Bedford, MA, 1973.

18. Biba, K. J., *Integrity Considerations for Secure Computer Systems,* ESD-TR-76-372, MITRE Corp., 1977.

19. Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December 1985.

20. Irvine, C. E., Anderson, J., Robb, D. A., and Hackerson, J., "High Assurance Multilevel Services for Off-the-Shelf Workstation Applications", Proceedings of the National Information Systems Security Conference, Crystal City, VA, pp. 421-431, October 1998.

21. Irvine, C. E., Levin, T., Wilson, J. D., Shifflet, D., and Pereira, B., "A Case Study in Security Requirements Engineering for a High Assurance System", to appear in Proceedings of the Symposium on Requirements Engineering for Information Security, March 2001.

22. Wilson, James D., *A Trusted Connection Framework for Multilevel Secure Local Area Networks,* Naval Postgraduate School Master's Thesis, Monterey, CA, June 2000.

23. Balmer, Steven R., *Framework for a High-Assurance Security Extension to Commercial Network Clients,* Naval Postgraduate School Master's Thesis, Monterey, CA, September 1999.

24. Bryer-Joyner, Susan, and Heller, Scott, D., *Secure Local Area Network Services For A High Assurance Multilevel Network,* Naval Postgraduate School Master's Thesis, Monterey, CA, March 1999.

25. Laurie, Ben, and Laurie, Peter, *Apache: The Definitive Guide, Second Edition,* O'Reilly & Associates, 1999, page 67.

26. The GNU Project Web Site page, http://www.gnu.org

27. The Apache Software Foundation, "Apache HTTP Server: Security Tips", http://www.apache.org/docs/misc/security_tips.html

28. Wainwright, Peter, *Professional Apache*, Wrox Press Ltd., 1999, page 391.

29. Wainwright, Peter, *Professional Apache*, Wrox Press Ltd., 1999, page 212.

30. World Wide Web Consortium, "The Word Wise Web Security FAQ", http://www.w3c.org/security/faq/www-security-faq.html

31. SecurityPortal, The Focal Point for Security on the Net, http://www.securityportal.com

32. Wainwright, Peter, *Professional Apache*, Wrox Press Ltd., 1999, page 308

33. Khare, R., and Lawrence, S., "Upgrading to TLS Within HTTP/1.1", Internet Engineering Task Force Request For Comment 2817, May 2000, http://www.ieft.org/rfc/rfc2817.txt

34. Apache – SSL, "Apache Secured by SSL", http://ww.apache-ssl.org

35. "mod_ssl, The Apache Interface to OpenSSL", http://www.modssl.org

36. OpenSSL, http://ww.openssl.org

37. Wainwright, Peter, *Professional Apache*, Wrox Press Ltd., 1999, page 359.

38. Housley, R., K., and Hoffman, P., "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", Internet Engineering Task Force Request For Comment 2585, May 1999, http://www.ietf.org/rfc/rfc2585.txt

39. Peterson, Larry L., and Davie, Bruce S., *Computer Networks, A Systems Approach, 2e,* Academic Press, 2000, pages 673-698.

40. Eads, Bradley, *Developing A High Assurance Multilevel Mail Server,* Naval Postgraduate School Master's Thesis, Monterey, CA, March 1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Cynthia E. Irvine 3
   Computer Science Department Code CS/Ic
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Dr. Geoffrey Xie 1
   Computer Science Department Code CS/Xg
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Mr. James P. Anderson 1
   James P. Anderson Company
   Box 42
   Fort Washington, PA 19034

7. Mr. David Shifflet 1
   Computer Science Department Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Ms. Evelyn L. Bersack 2
   3525 E. Cuervo Lane
   Yuma, AZ 85365

9. Carl Siel 1
Space and Naval Warfare Systems Command
PMW 161
Building OT-1, Room 1024
4301 Pacific Highway
San Diego, CA 92110-3127

10. Commander, Naval Security Group Command 1
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort Meade, MD 20755-6585

11. Ms. Deborah M. Cooper 1
Deborah M. Cooper Company
P.O. Box 17753
Arlington, VA 22216

12. Ms. Louise Davidson 1
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

13. Mr. William Dawson 1
Community CIO Office
Washington DC 20505

14. Capt. James Newman 1
N64
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

15. Mr. James Knoke 1
Wang Government Services Inc.
7900 Westport Dr.
McClean, VA 22102-4299

16. Mr. Michael Focke 1
Wang Government Services Inc.
7900 Westport Dr.
McClean, VA 22102-4299